CASSINI+

Cassini+ ASIC Specification

Rev 1.1
February, 2003

Part #: 100-7243-01

Products Rights Notice:

You understand that these materials were not prepared for public release and you assume all risks in using these materials. These risks include, but are not limited to errors, inaccuracies, incompleteness and the possibility that these materials infringe or misappropriate the intellectual property right of others. You agree to assume all such risks.

THESE MATERIALS ARE PROVIDED BY THE COPYRIGHT HOLDERS AND OTHER CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS (INCLUDING ANY OF OWNER'S PARTNERS, VENDORS AND LICENSORS) BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THESE MATERIALS, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Sun, Sun Microsystems, the Sun logo, Solaris, OpenSPARC T1, OpenSPARC T2 and UltraSPARC are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon architecture developed by Sun Microsystems, Inc. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X / Open Company, Ltd. The Adobe logo is a registered trademark of Adobe Systems, Incorporated. Part of the products covered by these materials may be derived from the Berkeley BSD systems licensed by the University of California. Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product described in these materials. This distribution may include materials developed by third parties who have intellectual property rights therein. Products covered by and information contained in these materials may be controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists may be prohibited.

Preface

This documents represents the Cassini+ ASIC specification. Cassini+ implements the MAC (Media Access Control), as well as a subset of the PHY (Physical) layer functions for Ethernet in the speed range of 10 Mbps to 1Gbps. Cassini+ is designed to interface hosts over a PCI bus.

Who Should Use This Specification

This specification is mainly intended for the ASIC, board, and software driver developers involved in Gbps Ethernet. It may also be of interest to architects and designers responsible for host PCI interface definition and development, as well as physical layer network interface developers.

Before You Read This Book

This book references a number of other documents. Readers are assumed to be familiar with the following:

- *PCI Local Bus Specification, Revision 2.1*
- *Reconciliation Sublayer and Media Independent Interface Specification, 802.3u-1995 or later*
- *Fibre Channel FC-PH Revision 4.3*
- *Fibre Channel - 10-bit Interface. Draft proposed X3 Technical Report, Rev 2.2 December 1995*

Organization

The First chapter includes a brief introduction of the architecture, features, and applications of the device.

Chapter 2 provides an overview of the hardware including major datapaths, clock domains, and basic theory of operation.

Chapter 3 presents the software overview including details on data structures and driver considerations.

Chapter 4 describes the Cassini+ PCI bus interface logic and modules which couple that to the TX and RX logic units.

Chapter 5 outlines the TX functional block.

Chapter 6 outlines the RX functional block.

Chapter 7 describes the MAC, provides information on the PCS, serialink and diagnostic LEDs.

Chapter 8 summarizes a number of the major signal interfaces affiliated with functional blocks.

Chapter 9 gives an overview of the interrupt structure and summarizes error handling.

Chapter 10 outlines the RAS features of the chip and also gives an overview of the testability features contained in the chip.

Chapter 11 is the programmers model, including the register maps and recommended programming practices.

Chapter 12 provides the pinout and pin description of the device, as well as packaging and mechanical information.

Contents

This Page Intentionally Left Blank

Figures

Tables

Introduction 1

1.1 Purpose of Document

The purpose of this document is to describe Cassini+ , Sun's second generation PCI based Gigabit Ethernet ASIC. The internal hardware architecture, the external hardware interfaces and the programmer's interface are presented here. Reader familiarity with PCI platforms and local area networks is assumed.

1.2 Requirements of a First Generation Gigabit Ethernet Network Interface

First generation Gigabit Ethernet (GbE) subsystems were first introduced in 1997 and 1998. Their goals were fairly straightforward. They essentially were:

- rapidly implement the subsystem to an in-flux IEEE 802.3z specification
- provide transmit and receive datapaths capable of providing 1Gb/s communications

GEM was Sun's first gigabit ethernet ASIC design. GEM is a high performing network interface operating at 10Mb/s, 100Mb/s and 1Gb/s link rates. It incorporated a 66 MHz PCI bus interface to host memory as one part of accomplishing the second goal. The chip realized its design goals and is one of the highest performing gigabit ethernet cards available.

1.3 Requirements of a Second Generation Gigabit Ethernet Network Interface

As network link rates have increased faster than memory system and CPU performance over the past several years, it has become increasingly apparent that proper overall system design is critical for applications to realize the performance potential that a gigabit speed network provides. Inefficient implementation of the host memory system or host I/O bus interface, non-optimal protocol stack processing, or an underpowered or overworked host CPU can readily limit network performance to substantially less than link rate. Second generation GbE subsystems strive to reduce system bottlenecks by efficient design of their interfaces to the host system and by repartitioning the network packet processing pipeline to allow the network subsystem to take on

tasks normally done by the host. The tasks undertaken include ones the subsystem can do faster due to processing power put into the subsystem specialized for that task. Also undertaken are tasks which can be done in parallel to the host CPU doing other work which allows overall increased system performance (if not both increased networking and overall system performance).

A second generation design will naturally include developments within the Internet community which are movements towards standardizing new features. At the present time these developments include recognition and correct handling of 802.3 and ethernet frames with VLAN tags and also features which help support applications which require bandwidth or latency assurances (i.e. quality of service). Thus, a list of second generation features should include:

- hardware parsing of layer 2 headers of incoming traffic to properly handle 802.3 and ethernet frames and VLAN tagged encapsulated packets and also lay groundwork for subsequent higher layer parsing
- hardware parsing of layer 3 headers of incoming traffic to identify and properly handle IPv4 and IPv6 packets and provide preliminary groundwork for layer 4 parsing
- hardware parsing of layer 4 headers of incoming traffic to provide special handling for TCP traffic given its prevalence
- efficient host interface design on both hardware and software fronts
- wide datapaths (even easier to realize with current process technology) to assure interconnect between modules does not bottleneck hardware performance
- hardware features to allow more of the incoming packet processing to be done by the network interface rather than the host
- mechanisms to allow distribution of incoming RX packets between multiple CPUs in an MP system
- Hardware support for Quality of Service features such as RSVP protocol
- mechanisms to provide Energy Star and Reliability, Availability and Serviceability (RAS) functions

1.4 Cassini+ Features

Cassini+ 's features can be classified into *System* and *Network* related features. *System* features affect the Cassini+ to host interface. *Network* features are functions that map to the Media Access Control (MAC) layer in the ISO model hierarchy, and in some cases even to the PHYSICAL (PHY) layer. The feature list identifies which are the new features introduced in Cassini+ and those which were present in its predecessor, GEM.

- System Features (from first generation)
  - 64/32 bit PCI, up to 66MHz speed
  - 3.3V and 5V signaling for universal PCI cards
  - Register slave and DMA Bus Master interfaces
  - Multiple packet buffering in built-in FIFOs
  - Manages host queues via Descriptor Rings
  - Supports DVMA as well as 64 bit DMA address space

- H/W assist for TCP checksum generation/checking
- Fcode ROM support
- Serial interface (bit banging)

- System Features New with Cassini+
  - PCI 2.2 compliant (first generation was PCI 2.1)
  - Interfaces to driver/system software through multiple descriptor ring based queues for RX and TX
  - RX free buffers of page size to provide greater memory use efficiency
  - TX logic provides RSVP protocol support with four classes of service
  - Hardware parses packet headers through Layer 4 to efficiently handle incoming TCP streams
  - Hardware introduced to reduce CPU overhead for packet header processing
  - Hardware provided to assist load balancing for RX packets amongst multiple CPUs
  - Header parsing at Layer 2 to handle 802.3 and ethernet VLAN tagged packets
  - Microprogrammable sequencer does parsing of packet headers allowing some feature enhancement possibilities and facilitates correction to parsing algorithm without silicon changes
  - IPv4 and IPv6 support
  - RAS support

- System Features New with Cassini+
  - 4 PCI's interrupts can be used
  - IPsec's AH/ESP encrypted packets are detected, stored in the system memory under newly added rx_free_descriptor_ring_2's control.

    The 2 new rx features are described in Chapter 6.4.

- Network Features (from first generation)
  - 10/100/1000 Mbps speeds
  - Half and Full Duplex modes
  - Frame Based Link Level Flow Control (IEEE 802.3x)
  - 8B/10B coding and link initialization
  - Network interfaces:

    Parallel interface software configurable as:

    Nibble-wide single ended MII interface for 100 Mbps

    Byte-wide GMII extensions for 1 Gbps

    10-bit interface for use with external 1Gbps SERDES devices

- Network Features New with Cassini+
  - 16 perfect MAC address comparison registers

1.5 Hardware Assistance for Receive TCP Streams

1.5.1 Copy Reduction on TCP Streams

In previous NIC implementations, incoming packets are DMA'd by hardware into a driver buffer. Later, the driver assigns ownership of the buffer to the kernel. The kernel **copies** the data portion of the packet into memory which eventually will be accessed by the application. The header portion of the packet remains in the kernel in a linked list structure. If the application is receiving a TCP transported message larger than 1500 bytes, the data will be divided into a stream of packets numerically sufficient to contain the data. Each packet goes through the process outlined above with successive data payloads concatenated to form the final message given to the application. Note that in such a case where TCP data is split into multiple packets, the kernel may elect to not notify the application of data availability until the last packet in the stream arrives with its data.

Given that TCP traffic is a very significant portion of the overall network traffic, Cassini+ is designed to handle received TCP data streams more efficiently. Cassini+ will inspect incoming packet headers and identify TCP streams or flows. Cassini+ monitors socket connections by keeping in a table IP source and destination addresses and TCP source and destination ports. When a connection is established a new entry is made in the table and when the connection completes it is removed. In order for a connection to remain in the table the TCP stream must be "well behaved" as defined later in this specification and other documents. For a well behaved TCP stream, Cassini+ will split the header from the data in a packet and place it into a header buffer which has links pointing to the surrounding headers. Cassini+ "reassembles" data payloads of packets in a TCP stream into a single buffer spanning as many pages in virtual memory as necessary. It DMAs the data portion of the packet into a reassembly data buffer and concatenates packet data with data from the prior packet of the stream. Once this reassembly is complete Cassini+ notifies the driver. Rather than copying data into the application's memory space the kernel page flips the data into the application space. Page flipping moves data faster than a CPU block copy operation if the amount of data to be moved is of quantity X exceeding a threshold level known by the kernel for that type of machine. The net result is a copy is eliminated by Cassini+ hardware extracting data from packets in the TCP stream and reassembling them into a buffer which can be page flipped rather than copied into user space.

1.5.2 Dynamic Packet Batching

The CPU overhead affiliated with packet processing is significant. One portion of the overhead affiliated with copying data payload is addressed by Cassini+ as described above. In Cassini+, "interrupt blanking" techniques are used to reduce the number of interrupts actually serviced by the CPU to less than one per packet. The interrupt reduction helps performance but it is still necessary for the CPU to process each header of each packet albeit more than one is sometimes processed at one interrupt.

Cassini+ extends overhead reduction for TCP streams by recognizing large messages contained in a TCP stream transported by a group of closely spaced TCP packets. Less CPU overhead will be incurred when using a protocol stack in the OS optimized to handle such a group of packets in conjunction with Cassini+ hardware. The hardware keys to this working successfully, however, are that

- batching is only over a sufficient number of TCP packets whose payloads have been reassembled into buffer as described in the previous section
- batching only occurs if the TCP packets being reassembled are sufficiently closely spaced such that round trip time calculations being performed by the protocol stack are not distorted due to excess delay of it being notified a packet has arrived.

Cassini+ meets these requirements with the logic it uses for packet reassembly and some modest extensions. Each time Cassini+ receives an incoming packet a post to the RX Completion Ring is made. The posting information contains “look-ahead” status indicating if the hardware has captured in any of the preceding stages of the logic up through the output of the RX MAC a packet which is part of the same flow being posted. These packets may be microseconds apart temporally but are spaced closely enough to not upset RTT calculations.

This process is referred to as “dynamic” packet batching because the look-ahead status placed into the RX Completion Ring is not only a function of packet timing but is also dependent upon the CPU’s ability to process packets per unit time. The faster the CPU is for a given set of TCP packets within a stream, the less batching will occur. The “look-ahead” is being done on contents within the RX FIFO which absorbs system response time. The more depleted the FIFO due to the system (which includes I/O bus and CPU processing) quickly processing incoming packets the less chance there is of a future packet of that TCP stream arriving before the current packet gets absorbed into the system. Conversely, if the system is bogged down and new entries on the RX Completion Ring are not being placed on the ring at the same rate as new packet arrival, the chances for a packet within the same TCP stream to enter the RX FIFO is increased. Given that the packet batching reduces the workload placed upon the host CPU, this mechanism is a tendency which helps relieve the system congestion.

1.5.3 Load Balancing

A third means of improving performance on receive traffic is to make use of multiple CPUs when present in the host platform to parallelize packet processing overhead amongst them rather than serialize it through one.

Cassini+ will do a hash on fields of the incoming packet and generate a six bit value which can be used by software to select one of N CPUs in an MP platform to do processing on the packet. The hash is done on IP and TCP header fields which causes all packets within a specific TCP data stream to be assigned to the same CPU and reduce any out of order processing introduced by load sharing.

Cassini+ load balancing can also work with non-TCP traffic. The hash is done on the Layer 2 (MAC) source and destination addresses which causes packets from a specific source to be assigned to the same CPU which, again, keeps out of order packet processing from being initiated by load balancing.

1.6 Software Dependencies

The performance features introduced in Cassini+ and described above require a new driver to exploit. In addition, software support for page flipping operation in the O/S in the machines targeted for Cassini+ is a requirement for TCP stream reassembly.

Dynamic packet batching requires support in the TCP protocol stack to optimize for closely spaced consecutively ordered packets within a flow.

Without these changes in the driver and pertinent portions of the kernel, Cassini+ will have no performance gain over previous generations of gigabit ethernet network interfaces.

1.7 Performance

It should be noted again that without changes/additions to the kernel and the development of a new Cassini+ driver, no performance gains will be realized.

The enhancements in Cassini+ over GEM are primarily intended to reduce host CPU overhead affiliated with incoming TCP traffic. Both Cassini+ and GEM are 1 gigabit per second network interfaces and for many situations very comparable performance will be observed from the two. Situations where Cassini+ performance can exceed GEM performance include:

- backbone servers handling large traffic loads
- client receiving TCP streams at high average data rate from server
- server receiving TCP streams at high average data rate from multiple clients
- dual or more processor machines receiving considerable network traffic (TCP or otherwise)
- applications with controlled latency or bandwidth requirements where RSVP protocol is implemented through network between source and destination

Many applications fall within this broad range of situations such as WEB servers and clients, file servers, backbone servers, and multimedia servers. TCP is a large part of Internet traffic making hardware designed to efficiently process it essential for second generation gigabit network architectures.

It should be understood that the clear performance benefit of all features outside of empirical measurements in a specific traffic environment is difficult to gauge. Performance features are individually enabled and feature inclusion in Cassini+ provides a hardware vehicle for the organization to assess how well each works in live situations.

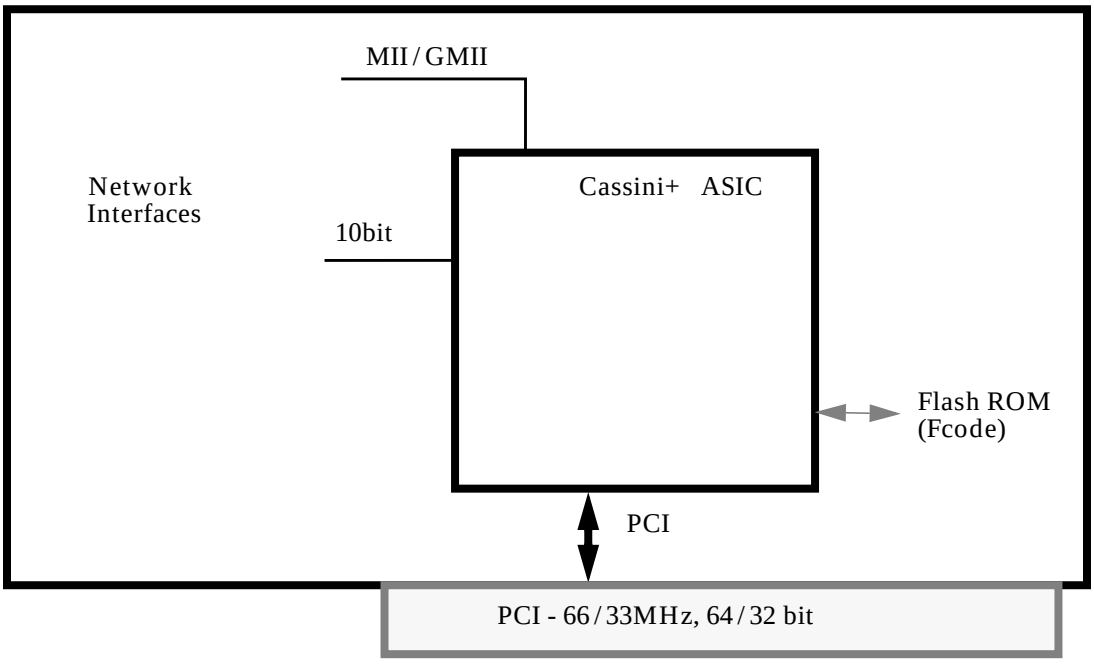
1.8 Cassini+ Interfaces

The Cassini+ ASIC has an electrical and a software interface model. These two are described briefly in this introductory chapter of the specification.

1.8.1 Basic Hardware Interface

The following diagram shows the basic interfaces to the Cassini+ ASIC.

Figure 1-1 Cassini+ ASIC interfaces



PCI - Universal voltage, up to 66MHz, 32/64 bit interface

MII/GMII - Media Independent Interface: 25MHz nibble wide/125MHz byte wide.

10 bit - 125MHz balanced code interface to external SERDES

Flash ROM - 8 bit interface to external Fcode Flash ROM

1.8.2 Basic Software Interface

1.8.2.1 Registers

The Cassini+ driver manipulates registers and memory elements within the slave interface of the ASIC. Programmed I/O operations are used during these accesses. The registers, their functions, and their addresses are described in the Programmer's Model chapter of this specification.

1.8.2.2 Host Memory Structures

Data to be transmitted, packets received from the physical layer, and data management structures are held within host memory. Cassini+ accesses these via D(V)MA operations and protocols are established with the host CPU that establish whether the owner of a host memory resource is the CPU or Cassini+ .

Cassini+ Hardware Overview 2

2.1 Overview

Cassini+ incorporates most of the functions required to implement a single-function Gigabit Ethernet PCI network interface. Figure 2-1 shows a top level diagram of Cassini+ 's functional blocks. The functions implemented in each block are:

1. PCI Interface
   a. PCI configuration space
   b. Slave Memory mapped I/O registers
   c. DMA bus master interface
   d. Arbitration between DMA channels
2. DMA engines
   a. One DMA channel for transmit and one for receive
   b. Descriptor ring DMA management
   c. 64 bit DMA address space
3. Transmit and Receive FIFOs
   a. Absorb bus access latencies and bus bandwidth variance
   b. De-couple of bus bandwidth from network functionality
   c. Generate Transmit TCP checksum
   d. Lossless flow controlled reception
   e. Prevent re-transmission and bad received frames from consuming system bus bandwidth
4. MAC
   a. Framing and data delineation
   b. CRC checking and generation

c. Unicast and Multicast Address Filtering

d. Flow Control MAC Frame reception and enforcement

e. Flow Control MAC Frame transmission

f. CSMA/CD protocol in Half Duplex

g. Independent Transmit and Receive functions in Full Duplex mode

5. PCS (Physical Coding Sublayer)

a. Encodes transmit data into 10-bit codes

b. Receive synchronization using 8B/10B sync character

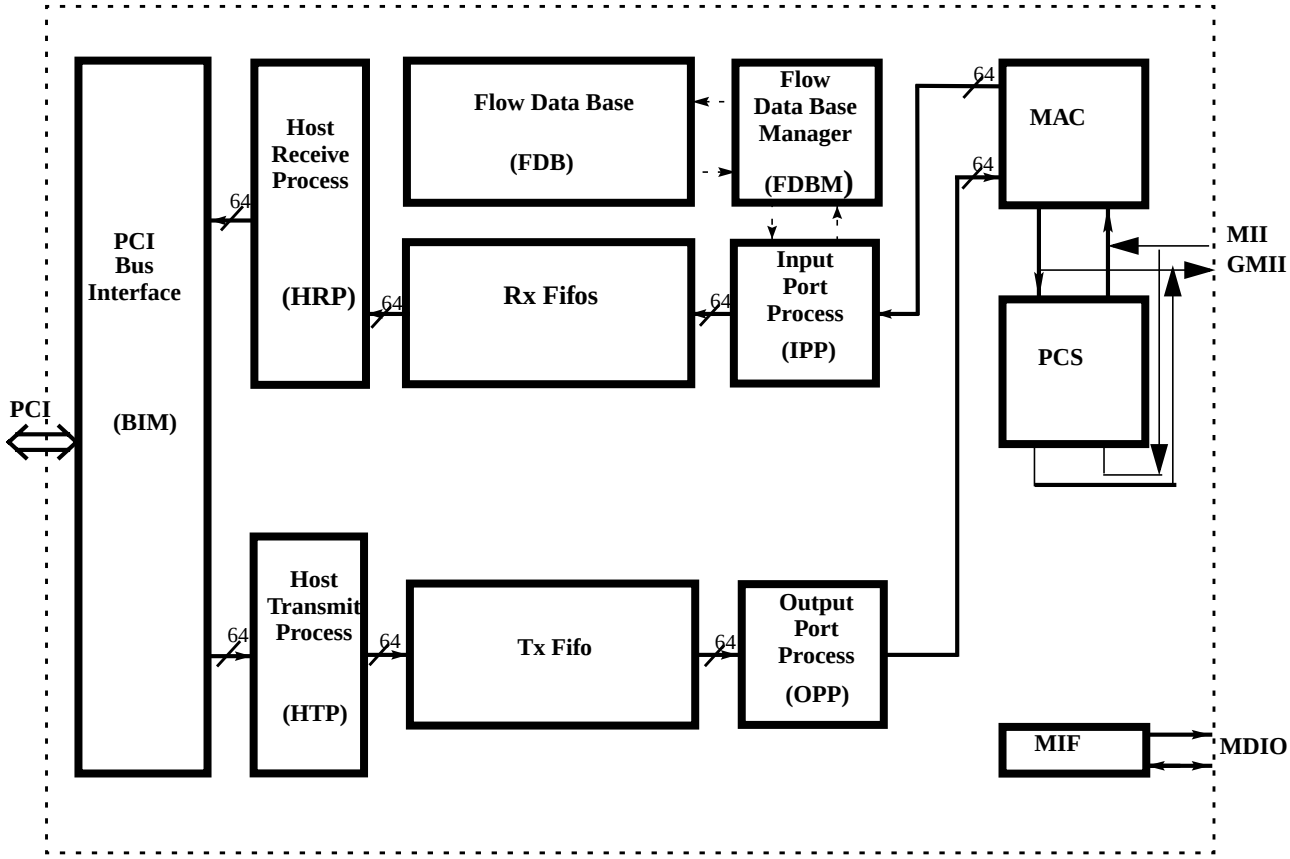
c. Decodes 10-bit receive data codes

d. Out of band signaling using special codes

e. Code error detection

f. Link Initialization

Figure 2-1 Cassini+ Functional Blocks



2.2 Hardware Overview

2.2.1 Terminology/Conventions

double-word - Used in this specification when refering to 64-bit fields in data structures or internal Cassini+ interfaces

frame - Stream of bits that flows between two nodes at the MAC layer. Includes the Preamble, the SFD, the Destination and Source Addresses , the Length/Type field, the Data/Pad bytes, and the FCS. All frame octets, except the FCS octets, follow a low order first format (least significant bit is transmitted first). The Length/Type field is transmitted with the high order octet first.

MAC address - 48 bit addresses respresented in this specification as 6 bytes of the form *a:b:c:d:e:f*. For globally administered addresses *abc* comprises the Organization Unique Identifier (OUI) part of the address. The least significant bit of *a* is the Multicast bit, and appears first in the media.

2.2.2 Clock Domains

Cassini+ 's clock domains are illustrated in Figure 2-2. As indicated in Figure 2-2, there are two major clock domains in the chip - 125 MHz for the core and 33 MHz / 66 MHz for the bus interface unless it is run at some slower rate in a low power consumption mode.

2.2.2.1 PCI Clock Domain

This clock is sourced by the PCI bus and is defined to operate in the range of up to 66.67MHz. It is used to drive the BIM block.

2.2.2.2 System Clock

This clock is sourced by an on-board oscillator and is defined to operate at 125 MHz. It is used to drive the Rx, Tx, BIM, SEB and MAC blocks.

2.2.2.3 Tx Clock Domain

This clock is used to drive the Transmit Protocol Engine in the TxMAC, TxPCS and Tx Serialink. In MII mode it is sourced by an MII external transceiver and is defined to operate at 2.5/25MHz +/-100ppm. For 1Gbps operation the TxMAC clock is a 125MHz byte clock. This byte clock is either the 125MHz TBC byte clock of the SERDES (internal or external) function, or the GMII transmit clock. The synchronization between this clock domain and the Local clock domain is performed across the Synchronization Fifo in the TxMAC.

2.2.2.4 Rx Clock Domain.

This clock is used to drive the Receive Protocol Engine in the RxMAC, RxPCS and Rx Serialink. In MII mode it is sourced by an external MII transceiver and is defined to operate at 2.5/25MHz +/-100ppm. For 1Gbps operation the RxMAC clock is a 125MHz byte clock. This receive byte clock is directly supplied to the RxMAC either by the SERDES (internal or external) function, or by the GMII interface. For external SERDES devices connected to the 8B/10B interface, the RxMAC clock is derived from the rising edges of the RBC[0] RBC[1] inputs. The synchronization between the RxMAC clock domain and the Local clock domain is performed across the Synchronization Fifo in RxMAC.

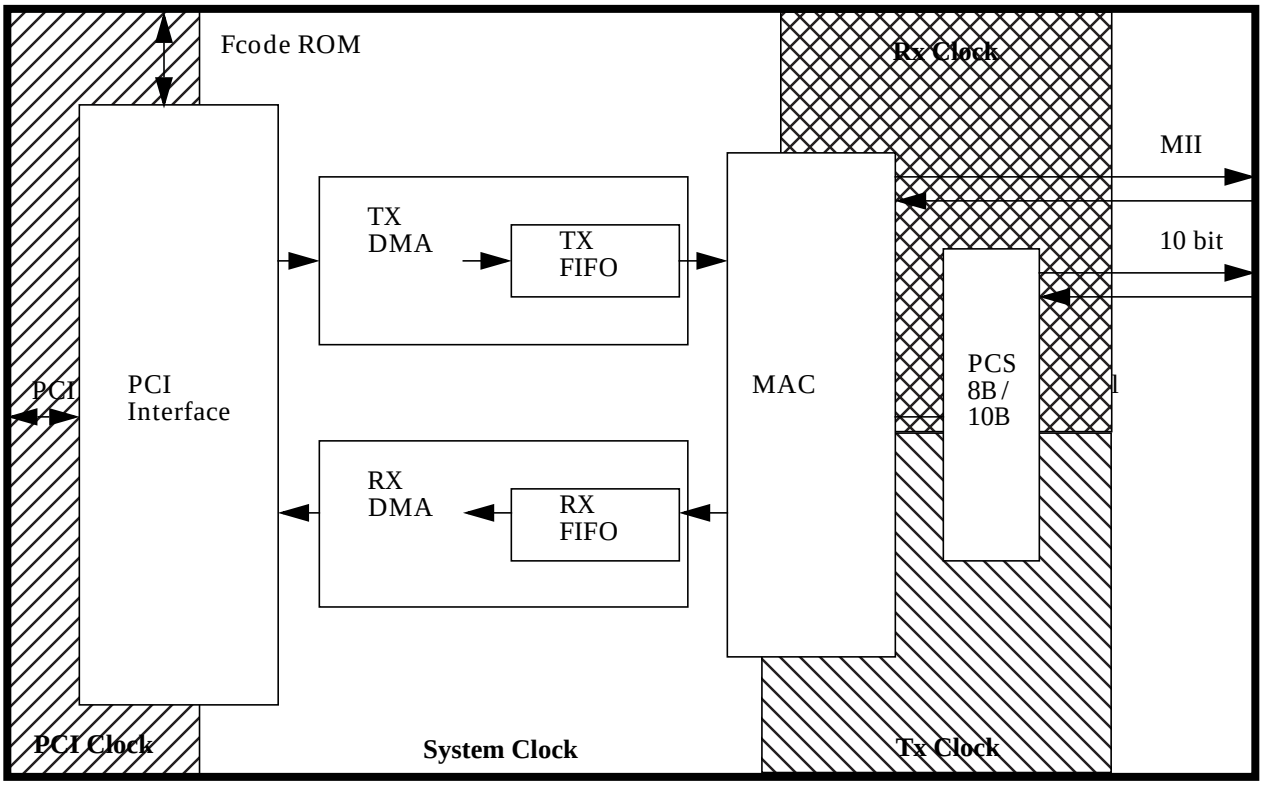


Figure 2-2 Clock Domains

2.2.3 Internal Data Paths

Internally there are two logically independent data paths, transmit and receive. These two paths meet in the PCI Block, as both paths must share a single PCI bus interface.

While the data throughput over the PCI and the network might be variable, the internal data paths between these boundaries are designed to sustain the peak throughput of these interfaces.

Most of the blocks in the data path contain some degree of data buffering, dictated by their own functional needs, with the bulk of the data buffering between the network and the host residing in the TX FIFO and RX FIFO blocks.

2.2.3.1 TX Data path

Figure 2-3 shows the width and speed of the TX Data Path between the different blocks.

Transmit data is transferred by the TX DMA block into the TX FIFO through the read buffers of the PCI interface block. The TX DMA and TX FIFO are capable of sustaining the maximum peak PCI burst rate for an entire packet transfer. The TX FIFO size is 9kbytes.

On the output side of the TX FIFO, data is transferred to the TX MAC using a 64- bit interface, nominally faster than the 1Gbps data rate of the network. The MAC layer, adds the relevant frame fields (Preamble, FCS) on the fly and generates a byte wide data stream (or nibble stream when using the MII).

The 8B/10B block encodes each byte into a ten bit code, which can be internally or externally serialized into the 1.25Gbps serial bit stream of an external serialink.

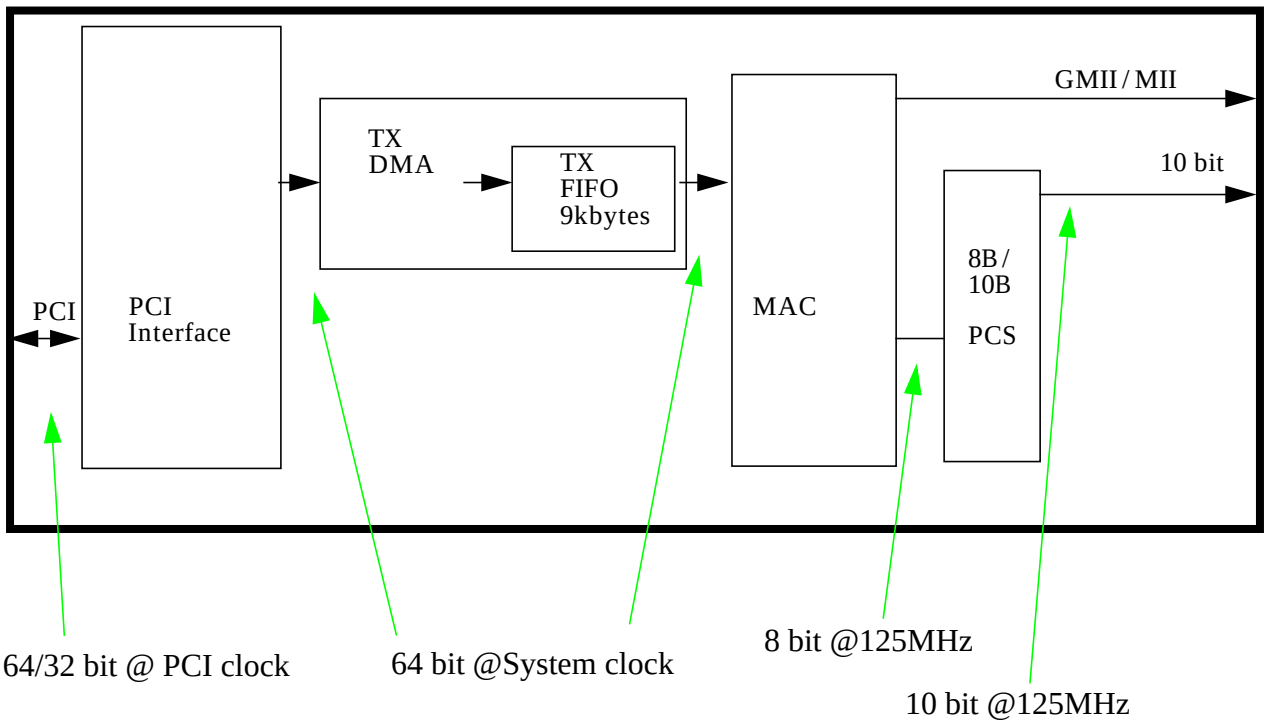


Figure 2-3 TX Data Path

2.2.3.2 RX Data path

Figure 2-4 shows the width and speed of the RX Data Path between the different blocks.

The 1.25 Gbps receive bit stream is presented either directly to the internal Serial Link module, or to an external SERDES chip. After clock recovery and bit alignment, the receive path is a 10 bit wide receive stream.

The 8B/10B block decodes it into a byte stream. The MAC is capable of receiving the bytes (or a nibble wide MII path) and after frame delineation, address filtering and FCS checking, pass 64-bit wide data, one frame at a time, to the RX FIFO. The RX FIFO size is 16 kbytes.

The RX DMA will move the data to the PCI interface write buffer, while sustaining peak PCI burst DMA rates for entire packets if necessary.

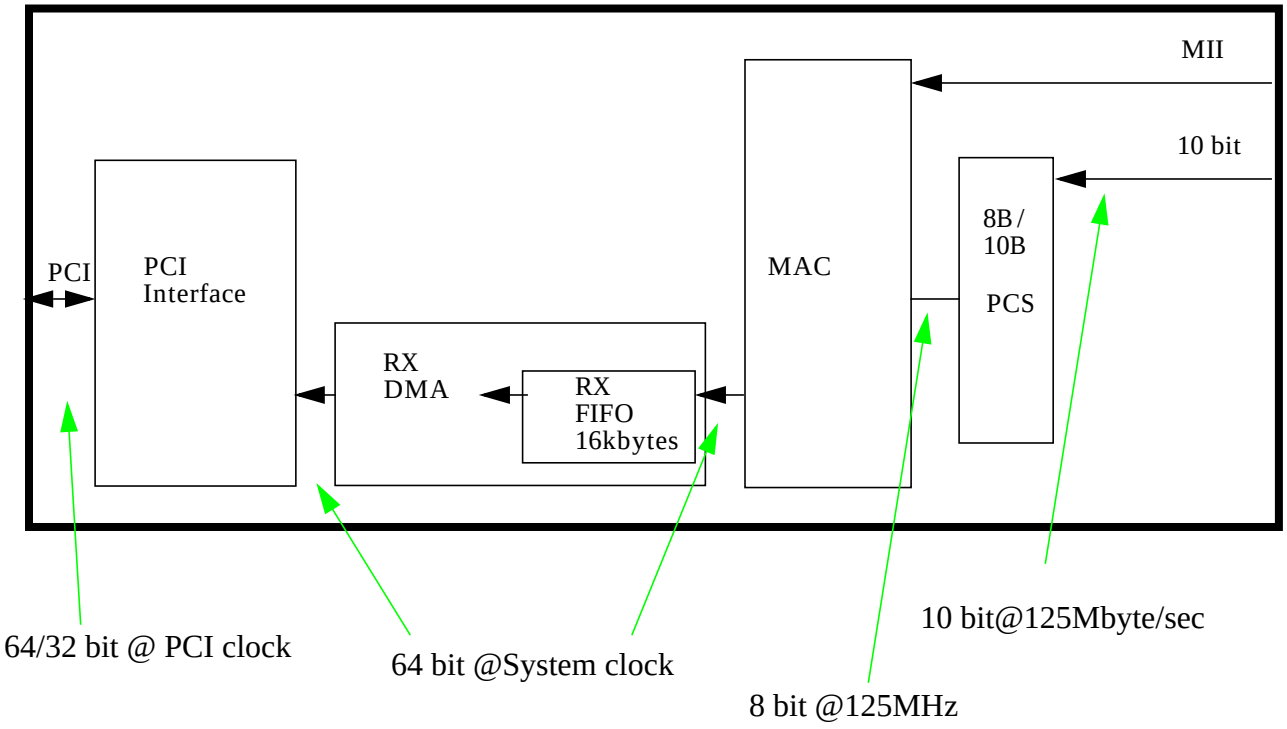


Figure 2-4 RX Data Path

2.2.4 MAC Control Frames

In addition to the usual Data frames, Cassini+ supports MAC Control frames, as defined by IEEE 802.3x, for the purpose of implementing link level flow control. MAC Control frames are typically sourced and terminated at the MAC layer, and do not traverse the entire data path to and from the host interface.

MAC Control frames are distinguished by a specific value in the frame *type field*. The specific MAC Control function is determined by the value of the MAC Control Opcode field.

2.2.4.1 Handling of Received MAC Control Frames

On reception, MAC Control frames are decoded and acted upon by the MAC block itself. In the particular case of flow control frames, the MAC Destination Address may be a well known group address or the unicast address, and the MAC Control Opcode field is "*PAUSE*". A MAC Control frame, like any other frame, must have a correct CRC to be considered valid.

Cassini+ detects a “*PAUSE*” flow control frame by matching the Destination Address, the type field, and the opcode field to pre-programmed values, as well as verifying correct CRC. The duration of the requested PAUSE, is extracted from the frame and internally stored.

Cassini+ ’s pause timer is loaded with the PAUSE value, at the first opportunity after receiving a PAUSE frame that there is no transmission in progress at the MAC. The MAC may not transmit Data Frames while the pause timer is not zero. MAC Control frames are not normally passed to the system, but the pause timer as well as the last PAUSE value received are readable by the host.

2.2.4.2 *Sourcing of MAC Control Frames*

Given that MAC Control Frames are valid frames, they can certainly be sourced by the host as regular data frames, however this is not practical for lossless flow control due to the processing and queueing delays involved, and the fact that such MAC Control Frames, generated as Data Frames, would be delayed themselves by a non-zero Pause Timer condition.

The appropriate sourcing of flow control frames is internal to Cassini+ . Cassini+ can transmit MAC Control Frames that bypass the data path (i.e. do not go through the TX DMA to MAC interface). These frames are assembled by the MAC block using programmable values for Destination Address, Source Address, Type Field, MAC Control Opcode, and its “associated parameter” field.

For “PAUSE” Frames the host should program these fields with the values specified by 802.3x, and the associated parameter with the desired PAUSE timer value. Cassini+ can autonomously send the pre-programmed PAUSE frame when the RX FIFO fills above a programmable OFF threshold. The PAUSE frame is emitted by the MAC at its next possible opportunity.

Whenever the RX FIFO usage falls below a programmable ON threshold, a similar PAUSE frame is emitted with the PAUSE timer value set to zero, allowing the sender to resume even if the original PAUSE timer had not expired yet.

Cassini+ ’s behavior is further constrained by the following guidelines:

1. After emitting an XOFF PAUSE frame, a second XOFF is not emitted unless the XON threshold was crossed.
2. XON frames will not be emitted unless the last PAUSE frame emitted was an XOFF.

Note – This alternating XON/XOFF scheme minimizes the number of PAUSE frames emitted. If the XON threshold is not crossed, it is possible for the sender to resume transmission and for the receiver not to emit additional XOFF PAUSE frames. For lossless performance the PAUSE timer value programmed in Cassini+ must be large enough to ensure that the XON threshold will be crossed.

These two types of PAUSE frames can also be emitted, upon host request, by using Cassini+ 's PIO write instructions. There are no restrictions on PIO triggered PAUSE frame emission.

PAUSE frames are triggered by the RX DMA block, but the enforcement of the above mentioned constraints, and the actual frame generation are done by the MAC according to the format shown in Table 2-1.

Table 2-1 Emitted PAUSE Frame Format

Field	Size	XOFF PAUSE frame	XON PAUSE frame
DA	6 bytes	Multicast Address defined for frame based flow control: 01-80-C2-00-00-01. Stored in MAC Address 8,7,6 Registers	
SA	6 bytes	Station Source Address, as stored in MAC Address 0,1,2 Registers	
Type Field	2 bytes	Value stored in MAC Control Type Register. Defined to be 0x8808	
MAC Control PAUSE Opcode	2 bytes	0x0001	
Associated Parameter	2 bytes	Pause Time in Slot Time units. (Pause Time is controlled via the Send Pause Command Register)	0x0000
PAD	42 bytes	0x00	
FCS	4 bytes	Frame CRC	

The PAUSE function is only specified by 802.3x for Full Duplex mode, and it is Cassini+ 's software driver responsibility to disable PAUSE frame generation for half-duplex operation.

2.2.4.3 *Frame Discard Policy*

Lossless flow control can be achieved as described in 2.2.4.2 as long as the sender implements flow control and the “PAUSE” emission threshold is programmed appropriately (typically up to two maximum sized frames may still arrive after the emission of “PAUSE”).

A packet discard policy is necessary for cases where receive data overflows may still occur, for example, in half duplex mode, or in full duplex if the sender does not implement the flow control option. The discard policy in regards to the receive queues C and D of Figure 2-5 is as follows:

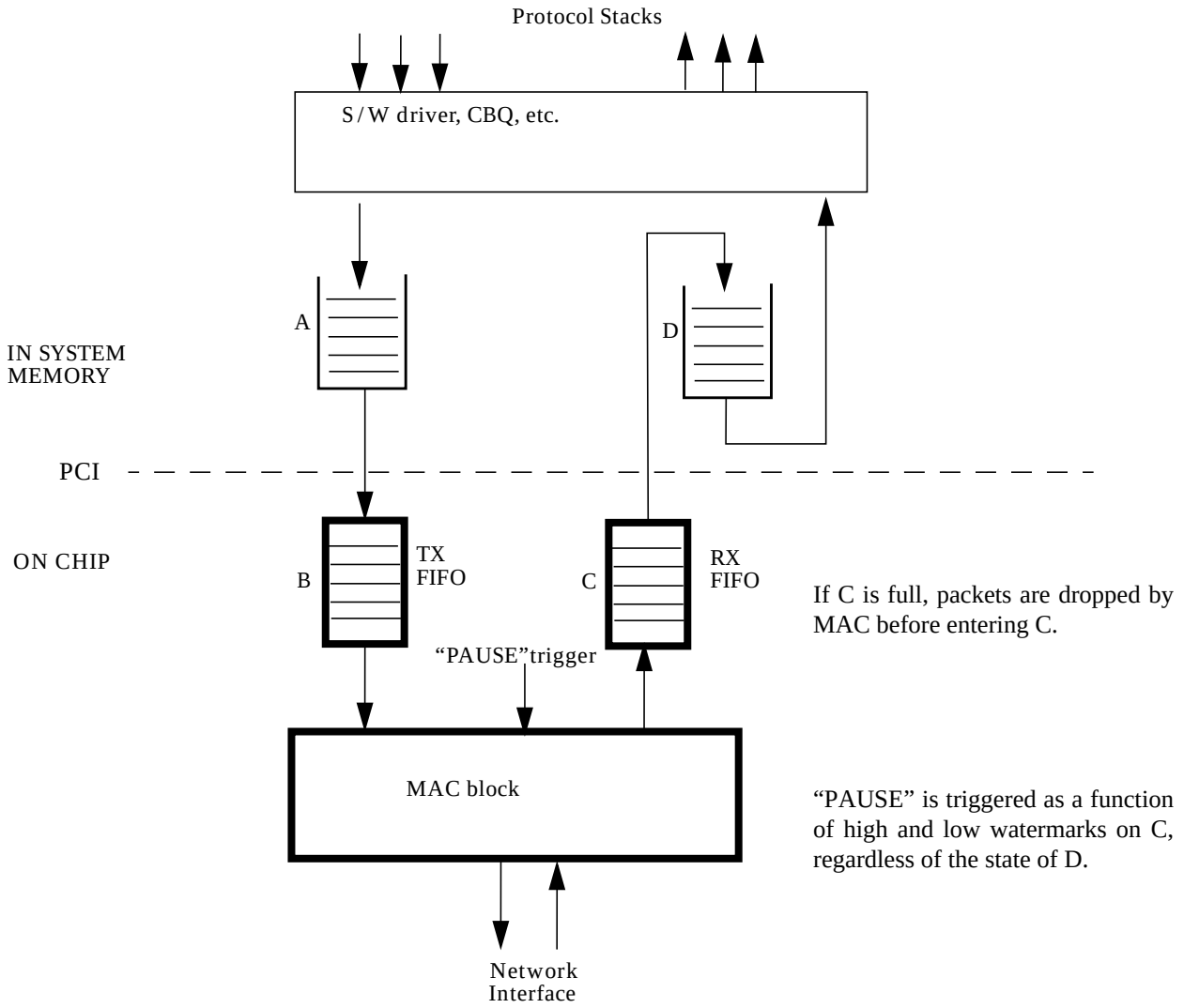


Figure 2-5 Frame Discard Policy

If queue D is full (no more free buffers in the receive system memory queue), Cassini+ can generate an interrupt to notify the driver, but does not drop data if there is still room in the RX FIFO C.

If C fills up (either due to D being full, or just due to transfers between Cassini+ and system memory being too slow), packet discards occurs at the input to C.

Once discard starts, it may only stop discarding on packet boundaries.

"PAUSE" frames are triggered as a function of C occupancy, regardless of whether D is full.

2.2.5 Network Interface

The network interface functions of Cassini+ provide the mechanisms for properly forming frames, following the protocols for when to send such frames, accepting frames, and allowing Cassini+ to present only the relevant frames and control information to the host. There is significant flexibility in the parameters that control these mechanisms, including network speed, access method, framing fields lengths and values.

It should be noted that all these network interface options have no impact on the data structures presented to the host, or the internal interface between the MAC and TX and RX FIFOs. Some of these options do affect the interface between Cassini+ and the external transceiver circuitry necessary to interface the network.

2.2.5.1 100BASE-T and 10BASE-T Networks

For 100BASE-T networks, Cassini+ preserves the functional partitioning at the MII level. Cassini+ implements all MAC layer functions above the MII in half and full duplex modes. 100Mbps (as well as 10Mbps) connectivity can be implemented using commercially available MII transceiver chips.

2.2.5.2 Gigabit Networks

The natural interface to the anticipated Gigabit links, in particular for fiber transceivers, is based on leveraging Fiber Channel type transceivers. Both FC-0 (Physical) and FC-1 (Code) Fiber Channel are applicable for Gigabit fiber links, but the leverage is directed mostly at using FC-0 devices, while incorporating the FC-1 function on chip.

Cassini+ provides a built-in PCS (Physical Coding Sublayer) with its corresponding 8B/10B coding. The PCS functions may be used in conjunction with external SERDES chips or Fiber Channel transceivers via a standard 10-bit interface, The external interface in this case is a serial PECL differential pair at 1.25Gbps.

When using external PMA and PMD components, note that ANSI X3T11 specifies 1.0625 Gbps operation, and Cassini+ requires transceivers capable of 1.25Gbps bit rate.

An additional interface is the GMII. It represents an extension of the MII to accommodate 1Gbps operation for non-8B/10B media interfaces.

In order to minimize the pinout impact of these interfaces, the MII, GMII, and SERDES interfaces share many pins, and typical Cassini+ boards should only populate one interface.

2.3 Data Structures

2.3.1 System Memory Data Structures

The transmit and receive queues in system memory are implemented as “wrap-around descriptor rings”. These queues are jointly maintained by Cassini+ and by the device driver. Given that the host frame processing time is likely to be dominant, and highly variable, large host queues are desirable in the descriptor rings. The number of entries is programmable in binary increments, from 32 to 8192 entries. Each descriptor consists of two double-word entries: a control/status entry and a pointer to a data buffer.

The interaction between the hardware and the software is managed using on-chip registers, a dma location for tx completion information as well as the content of the descriptors themselves. Recognizing that several descriptors may occupy a cache line, and that descriptors typically reside in consistent memory space, Cassini+ implements internal caching on descriptor reads to minimize the PCI and host overhead of repeated partial line reads and writes. Descriptor caching on reads applies to transmit and receive, and consists of fetching and internally storing up to four descriptors sharing a cache line. This has to be complemented by the software driver posting descriptors in groups of four, whenever possible. The equivalent technique for descriptor writes is RX completion entry write “batching” and is described in the TX & RX chapter.

2.3.2 Transmit Descriptor Ring

Figure 2-6 shows Transmit Descriptor Ring organization. Packets to be transmitted are placed into buffers in memory. Depending upon the class of service the packet is to be sent under, a descriptor characterizing the packet and pointing to the buffer is placed on one of four rings, each ring dedicated to one of the four quality classes of service defined in the RSVP protocol. Multiple packets are queued by placing them in buffers and placing one descriptor per packet on the appropriate descriptor ring. Cassini+ will start transmission of the packet when the driver writes into the TX Kick Register a value indicating which descriptors should be fetched by Cassini+ . The ASIC then reads in the buffers pointed to in memory containing the packets to be transmitted. After the packet is transmitted, Cassini+ writes to the memory based Completion Register associated with the descriptor ring a value signifying the descriptor whose associated packet has been transmitted. This allows the driver to reuse the memory buffer and descriptor ring entry for subsequent transmission packets. The figure below illustrates these structures.

Again, full details on transmit descriptors including field definitions are provided in the Cassini+ Software Overview. Additional detail can also be found in this document in the Programmer’s Model section.

For Transmit, descriptor caching reduces bus overhead by reading up to four descriptors in the same burst, however the software driver cannot guarantee that it will post more than one descriptor at a time.

The TX DMA may only internally cache descriptors if the TX Kick Register indicates that there is more than one descriptor to be fetched within the same cache line. Otherwise a single descriptor is read.

The maximum number of outstanding transmit descriptors that Cassini+ can handle is given by the transmit descriptor size minus one.

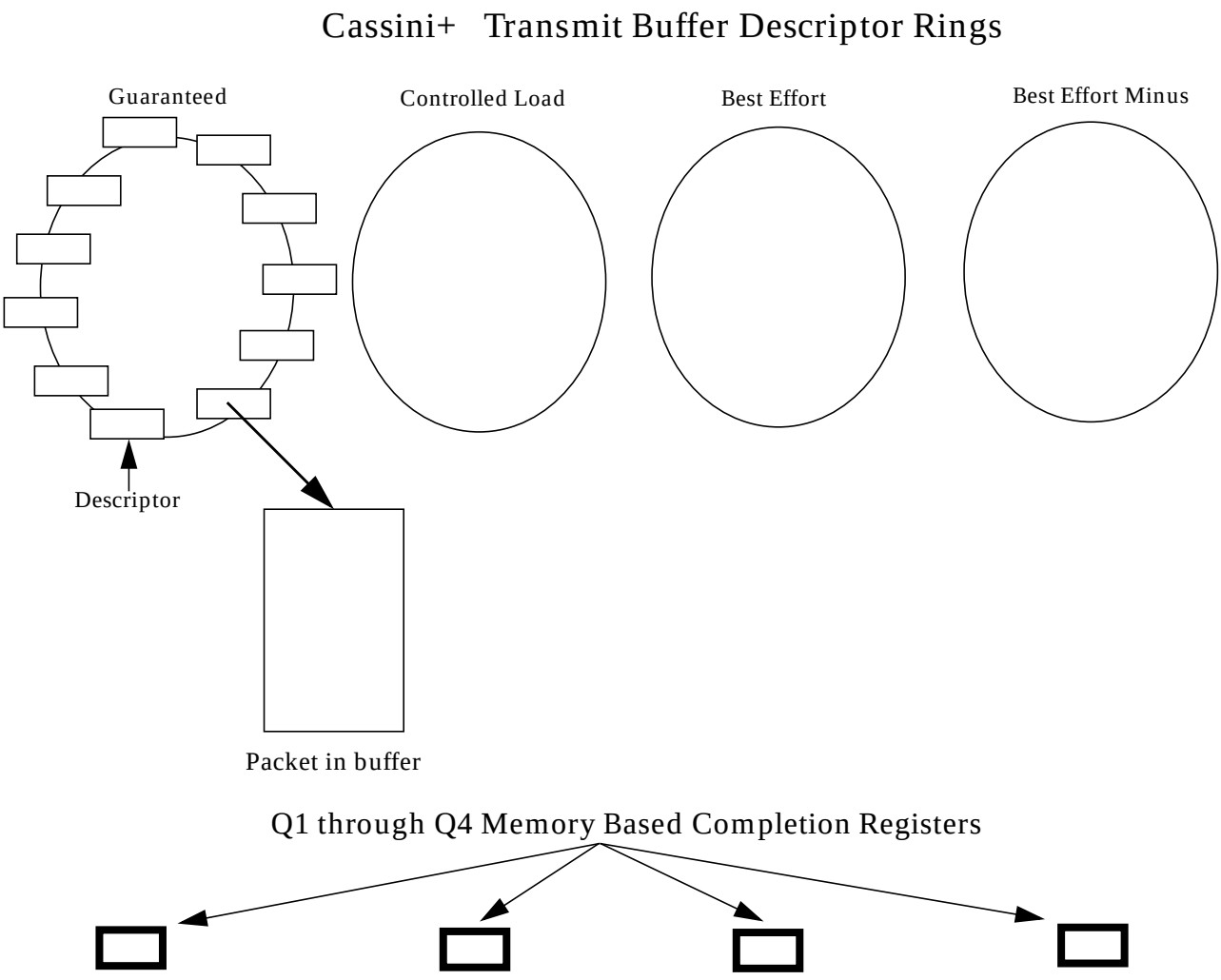


Figure 2-6 Cassini+ Transmit Host Memory Structures

TX Descriptor Entry Layout

63 34	33	32	31	30	29	28 21	20 15	14 0
RESERVED	No CRC	Int Me	Start of Frame	End of Frame	Checksum Enable	Checksum Stuff Offset	Checksum Start Offset	Buffer Size
Data Buffer Pointer								

All TX Descriptor fields are written by the host and read by Cassini+ . Cassini+ never writes into TX Descriptors.

2.3.3 Receive Descriptor Ring

Figure 2-8 shows the Receive Descriptor Ring organization.

Incoming packets must be directed into available memory buffers expeditiously to avoid RX FIFO overruns and subsequent packet loss. The RX host memory structure consists of a descriptor ring containing descriptors pointing to free buffers an incoming packet can be DMA'd into by Cassini+ . It also has an RX Completion Ring so Cassini+ can inform the driver which buffers and descriptors have been utilized and are now ready for software processing.

Most prior ethernet controllers using a free buffer descriptor ring architecture only had a single size buffer pointed to by the descriptor. The buffers pointed to by those descriptors needed to be a maximum of the network MTU in size which is 1500 bytes for ethernet networks. Small packets, unfortunately, would consume the entire buffer even though they might need only a few hundred bytes space. Cassini+ makes more efficient use of memory by using a page size buffer which can be assigned to one of four categories: 1) headers and small packets no more than 256 - swivel_offset bytes, 2) standard ethernet packets not being reassembled larger than 256 - swivel_offset bytes, 3) packets undergoing reassembly, and 3) jumbo packets, packets of size 1522 bytes to 9000 bytes. A page size buffer is used to collect small packets less than 257 bytes in size. Multiple small packets, up to [page_size/256], can be stacked into one of these "header" buffers. Given small packets are frequent occurrences in the network this mechanism saves measurable memory space and descriptors. Also, a page size buffer can be allocated for up to [page_size/1500] larger packets not part of a reassembly stream. Page size buffers are used to reassemble (i.e. concatenate) TCP streams that were split into packets. The reassembled stream will be at least 1500 bytes and if larger will take a bigger fraction of the page size buffer. Should the stream be larger than a page, another page sized buffer from the free buffer ring will be used to continue the reassembly. Finally, up to two page size buffers are used to store jumbo packets. The hardware architecture has no fixed limit on the number of pages that can be used to reconstitute the stream.

The first incoming packet which is part of a TCP stream being reassembled might access a descriptor and affiliated page size buffer prior to a small incoming packet which uses a subsequent descriptor on the descriptor ring and a different affiliated page size buffer. If a number of small packets arrive prior to the last packet of the reassembling TCP stream, the data buffer for the small packets gets handed to the driver prior to the buffer used by the TCP stream demonstrating the fact buffers will not always be returned to software in the order taken from the ring. Thus, a completion ring structure is used instead of a simpler completion register as employed by GEM or even Cassini+ 's TX. The basic structures are shown in the figure below.

Full details on receive descriptors and the receive completion ring including field definitions are provided in the Cassini+ Software Overview. Additional detail can also be found in this document in the Programmer's Model section.

If receive packets arrive far apart Cassini+ updates the RX descriptors one at a time (cacheline write). If receive packets arrive at a higher rate, Cassini+ will update two descriptors at a time, writing entire cache lines.

Given that the software driver must post descriptors in multiples of four, and that the ring should not wrap around, the number of outstanding descriptors cannot exceed the descriptor ring size minus four.

Each packet written into host memory is followed by a post of a 32 byte descriptor on the RX Completion Ring as governed by the HRP control logic. (Some cases actually result in two posts as described later in this document.) The fields are fully defined in the Cassini+ Software Overview chapter.

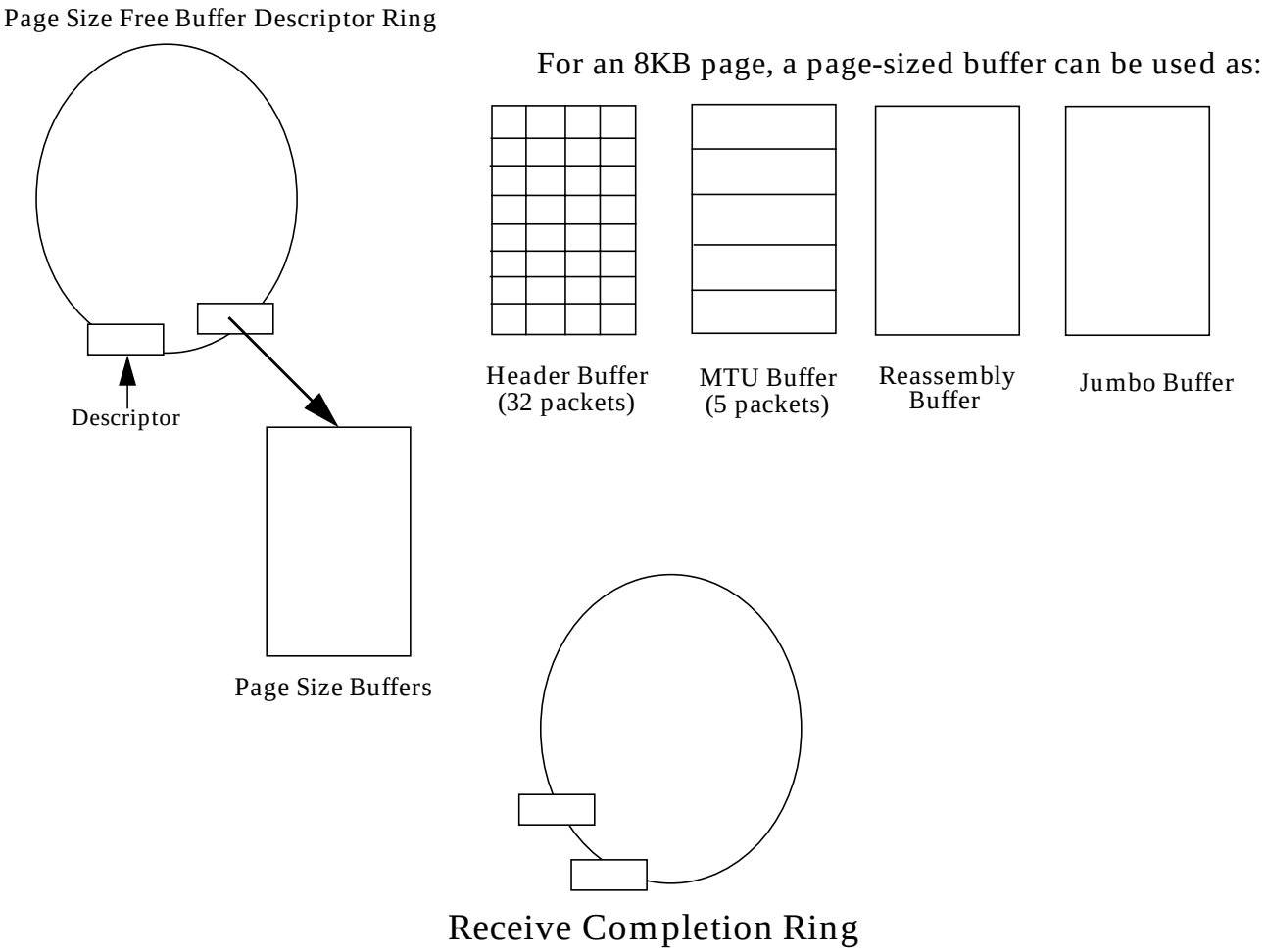


Figure 2-7 Cassini+ Receive Host Memory Structures

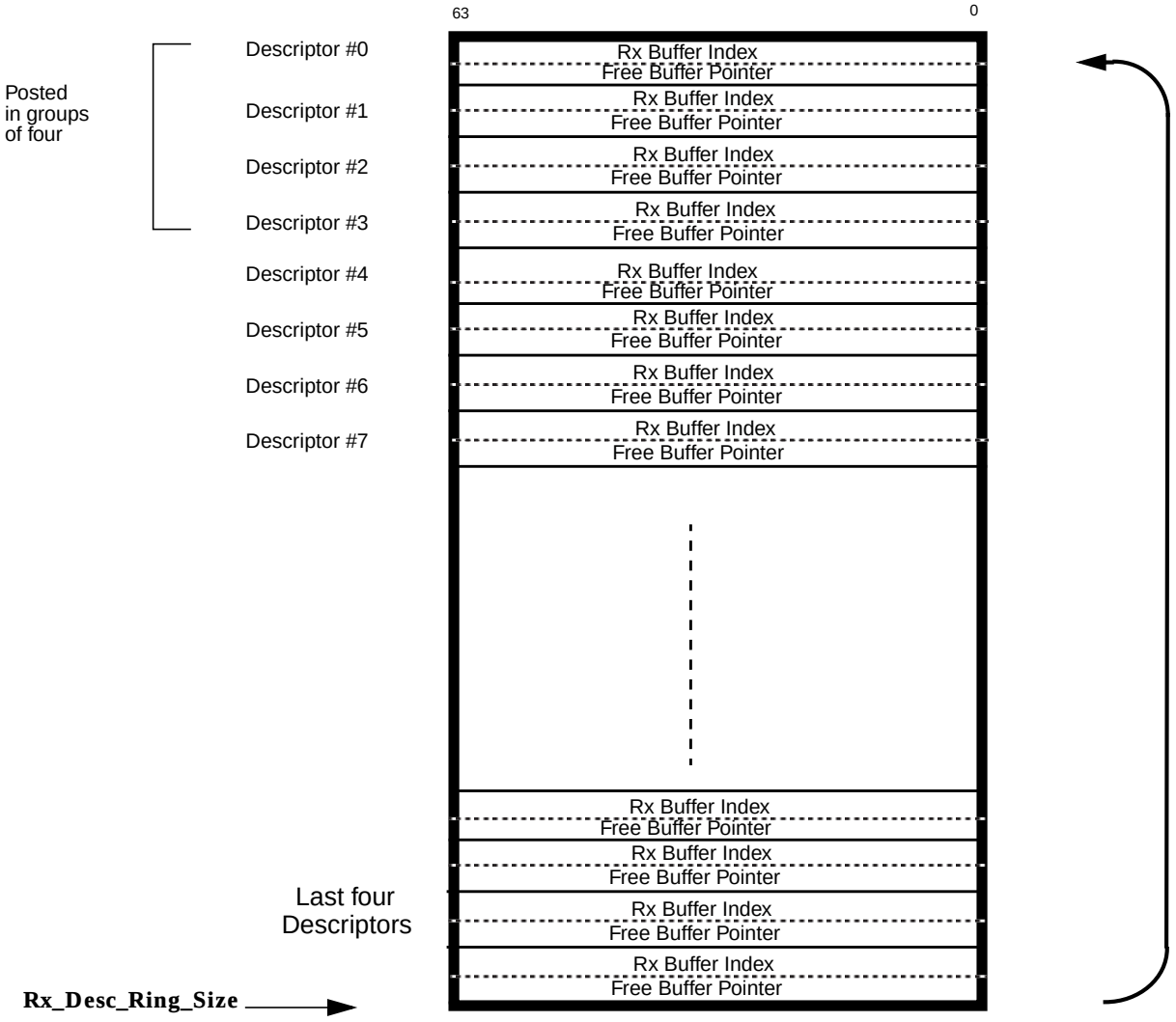


Figure 2-8 Receive Descriptor Ring Organization

2.4 Theory of Operation and Data Flow

2.4.1 Frame Transmission

Host Memory to TX FIFO Data Transfer Process

1. Frame transmission is initiated by an upper layer protocol posting a frame to a transmit queue in host memory. The frame may reside in one or more data buffers. The transmit queue could be any one of the four classes of the descriptor rings described in Figure 2-6.

2. The device driver allocates a descriptor for each buffer in the corresponding transmit descriptor ring.

3. The device driver writes into the corresponding TX Kick Register the descriptor number that follows the last valid descriptor.

4. Upon detection that there is at least one descriptor in any of the four queues, evident from the fact that the value of the TX Kick Register is ahead of the corresponding TX Completion Register, the TX DMA arbitrates for access to the PCI bus. The arbitration between the TX and RX DMA is done by the PCI Interface block. Note that if more than one descriptor ring has buffers queued, then the TX QUEUE Arbiter (described in the TX & RX chapter) decides which queue is to be serviced at that time.

5. The TX DMA engine uses a DMA burst read and fetches a number of descriptors given by MAX [valid new descriptors in the cache line , 4]. The descriptors read are cached internally for use in the order they were fetched.

6. The TX DMA starts transferring the contents of the corresponding host data buffer into the TX FIFO, and advancing the Write pointer accordingly. The DMA read data transfer attempted is always a burst of an integer number of cache lines, except possibly at the end of the frame transfer if less than a full line remains.

7. On its way to the TX FIFO, the data passes through the Chaining block where byte alignment is performed. For the first host data buffer of a frame, the data is aligned to a double-word boundary in the TX FIFO. For subsequent buffers, “byte rotation” is performed to form a contiguous data stream in the TX FIFO for each frame.

8. As the “chained” data stream “flies by” on its way to the TX FIFO, the Checksum block monitors it, and the TCP checksum is computed starting from the “start_checksum_offset” byte. Also, the word indicated by the “stuff_checksum_offset” is saved in a temporary holding register.

9. Upon completion of a host buffer transfer, the TX DMA engine turns over the descriptor ownership back to the device driver by writing into the TX Completion Register the descriptor number that follows the last descriptor processed. This value of the TX Completion Register is also written into host memory at location indicated by or offset from value in TX Completion Writeback Address Registers Low / high.

10. If the frame contains more than one host data buffer, and descriptors are already cached internally, steps 6 through 9 are repeated until the entire frame has been transferred from the host memory to the TX FIFO. If at any point there are no more cached descriptors for the frame, steps 4 through 9 are repeated instead.

11. If the TCP checksum generation for the frame is enabled, the computed checksum is stored into the holding register, and the updated word is loaded in the frame header in the TX FIFO using a programmable offset from the beginning of the frame as reflected by the Shadow Write pointer.

12. The TX DMA assembles a Control Word and appends it to the end of the frame in the TX FIFO. The Control Word indicates the last byte boundary of the frame, and indicates to the TxMAC whether to generate CRC for the frame. The TX FIFO Write Pointer is loaded into the Shadow Write Pointer and point to the beginning of the next frame in the TX FIFO.

13. Two interrupt mechanisms can report frame transfer completion at this point. The TX_DONE interrupt status bit is unconditionally set, and the TX_INT_ME status bit is set only if requested in any of the descriptors of the frame.

14. The TX FIFO packet counter increments for every frame written into the TX FIFO.

TX FIFO to Network Data Transfer Process

1. This process is triggered by the presence of at least one full frame (TX FIFO packet counter is not zero) in the TX FIFO, or by the number of bytes exceeding a programmable threshold. The TX FIFO read pointers (Read Pointer, Shadow Read Pointer) point to the beginning of the frame in the TX FIFO.

2. On the output of the TX FIFO, the frame is unpacked into a 64-bit data stream, and is transferred to the TxMAC in bursts of 32 bytes. The Read Pointer is incremented reflecting the last TX FIFO location read.

3. Should the TX FIFO be empty during frame transfer, data transfer to TXMAC stalls until more data becomes available in the TX FIFO.

4. The TX DMA relies on the TxMAC for discriminating between normal and late collisions. Upon normal collisions, the frame is re-transmitted by loading the Shadow Read Pointer into the Read Pointer. No retransmission is requested by TxMAC upon late collisions.

5. TX DMA assumes that normal collisions cannot occur after having transferred 1kbytes. Beyond that point the Shadow Read Pointer tracks the Read Pointer.

6. When the last data word (distinguished by the tag bit set) appears in the TX FIFO, the following word (status word, with the tag bit set as well) must also be examined to determine how many bytes are valid in the last word, and encode that information during the last data transfer to the Tx MAC.

7. The TX FIFO packet counter decrements for every frame read from the TX FIFO when the TxMAC requests the next packet. This guarantees that the current packet does not need retransmission.

2.4.2 Frame Reception

Network to RX FIFO Data Transfer Process

1. When a frame is received and it satisfies the RxMAC filtering criteria, the frame will be passed by the RxMAC to the IPP FIFO. The data is packed into 8-byte words by the RxMAC. Transfers take place whenever the RxMAC has at least 8 bytes to transfer and the IPP FIFO has sufficient room.

2. Before packet reception, the IPP Write Pointer and Shadow Write Pointer coincide and point to the next IPP FIFO write location. When a frame is being received into the FIFO, the Write Pointer increments while the Shadow Write Pointer continues to point to the beginning of the frame. If the abort bit is set in the status word from the MAC for that packet, the Shadow Write Pointer will be used to “rewind” or erase the packet from fifo memory.

3. While the packet is being stored in the IPP fifo, up to 114 bytes of the frame is also being stored in the Header Parser where it will be parsed. If the packet size is less than 54 bytes, the packet will require no Layer 3 or 4 hardware assist.

4. When the Header Parser has finished parsing the data, it will give results to the IPP. The IPP waits for this information in order to start transferring the packet to the Rx FIFO.

5. As the frame data stream “flies by” on its way to the RX FIFO, the Checksum block monitors it, and the TCP checksum is computed on the entire MAC frame, starting from the checksum_start_offset and for the length given by the TCP_Payload_Length.

6. If the Header Parser has determined the frame will receive Layer 3 and 4 hardware assist, it will make a request to the FDBM for flow information.

7. Using the key, sequence number and control information provided by the Header Parser, the FDBM will determine if there has been a flow match and provide an opcode to be used eventually by the HRP. This information is used by the IPP to form a control packet.

8. After the entire frame has been transferred to the RX FIFO, the status word from the RxMAC is modified to include, among other things, the TCP checksum. The frame length is provided by the Rx MAC as part of the status word. The last word as well as the status word are written into the RX FIFO with the tag bit set.

9. The Rx Load Control subsequently compiles a control packet consisting of the opcode, TCP payload offset and size and adds it to the control fifo. The presence of a packet in the Rx Control FIFO will trigger action by the Rx Unload.

10. If at any point during this process the RX FIFO occupancy rises above the OFF threshold, a “XOFF PAUSE” frame transmission request to the Tx MAC may be triggered, as described in Section 2.2.4.2, “Sourcing of MAC Control Frames”. The Tx MAC may ignore the request if it is not configured appropriately (Send_Pause_Enable must be set).

RX FIFO to Host Memory Data Transfer Process

1. If at least one descriptor is internally cached when the RX DMA needs to move Data to Host Memory, it proceeds as described in step 3. If no descriptors are internally cached it proceeds with step 2.

2. If the values of the RX Kick register and the RX Completion register become equal, the RX DMA generates an interrupt (Rx_Buf_NotAv). When the values of the RX Kick register and the RX Completion register are different and when the eight descriptor cache is at least half empty, the RX DMA engine fetches the next four descriptors from host memory using a DMA burst read, and proceeds to step 3.

3. RX DMA starts transferring the frame data from the RX FIFO to the host buffer when there is at least one packet in the RX FIFO.

4. For the initial transfers of a frame requiring reassembly, the RX DMA transfers the header to the header buffer and writes a completion descriptor. Then it transfers the data portion to a page-size buffer. If the packet is a non-reassembly packet it will be transferred in its entirety to a 256B header buffer or an MTU-size buffer depending on its size.

5. The frame transfer is completed when two consecutive tag bits have been detected by the RX DMA. If the status word indicates the packet was valid the RX DMA engine updates the registers that hold the descriptor contents with the appropriate values for the frame and clears the OWN bit.

6. Up to two sets of descriptors will be internally stored before an actual descriptor write is initiated on the bus. The descriptor number, and the RX FIFO packet counter are used to decide whether to update a descriptor immediately or to collect the second descriptors and then update both (“descriptor batching”). Descriptors are written whenever the second descriptor is ready, or the RX FIFO packet counter indicates no more packets await in the RX FIFO. The skip_num_desc field in the completion descriptor will indicate the number of 32B slots used for padding the descriptor write to the end of the cacheline.

7. After using any descriptor for a new packet and updating it in memory, an interrupt is generated to indicate to the device driver the successful completion of the frame transfer. This may be blanked for a period of time specified in the Interrupt Blanking Register.The value of the RX Completion Register is incremented by the number of Descriptors written (modulo RX Descriptor Ring Size) prior to setting the interrupt.

8. If at any point during this process the RX FIFO occupancy falls below the ON threshold, a “XON PAUSE” frame transmission request to the Tx MAC may be triggered, as described in Section 2.2.4.2, “Sourcing of MAC Control Frames”. The Tx MAC may ignore the request if it is not configured appropriately (Send_Pause_Enable must be set).

Cassini+ Software Overview 3

3.1 Overview

Cassini+ implements several advancements over previous ethernet designs at Sun.

On transmit Cassini+ provides multiple descriptor rings with an arbitration scheme to allow multiple qualities of service to be pushed down to the hardware level.

For receive Cassini+ implements three innovations expected to increase performance: reassembly of buffers to page sizes, batching of multiple packets, and hardware support for dispatching to multiple CPUs.

3.2 Driver Transmit Path

3.2.1 Multiple Descriptor Rings

Cassini+ 's most notable feature on transmit is support for multiple transmit descriptor rings, allowing multiple priorities and qualities of service for outbound traffic.

Allowing mblks from a single STREAMS queue to be sent to any of the four descriptor rings would pose difficulties in flow control. If one of the four descriptor rings fills, forcing the driver to begin queueing mblks in the stream, the other three rings will also feel the effects of the flow control. The driver would be unable to maintain the QoS for all four rings if all four come from a single queue. Therefore the driver will map each stream to one of the four queues, and thus requires separate stream for each ring. It is assumed that this will be implemented in a module placed between the driver and the protocol module, which has a private interface to the protocol layer for flow control to an individual destination.

A typical plumbing in this environment would be:

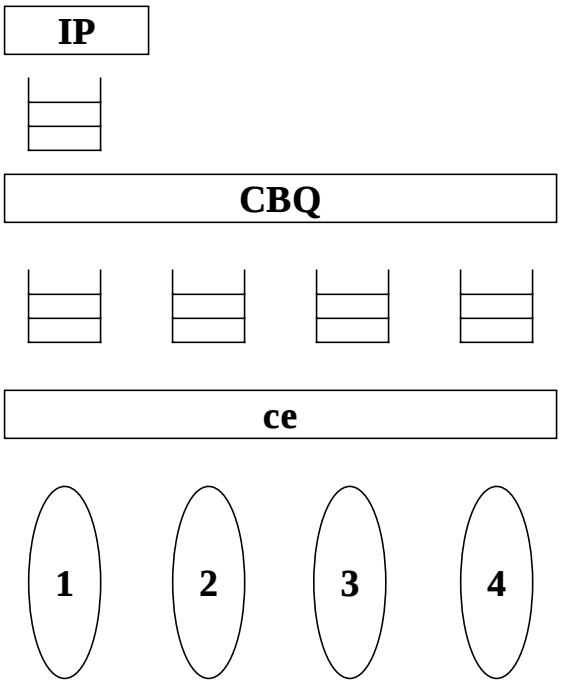


Figure 3-1 Typical STREAMS plumbing for multiple transmit rings

3.2.2 Transmit Descriptor Handling

Each stream coming in to the driver will contain a pointer to the data structures for the descriptor ring used by that stream. The driver will have four such data structures. The mechanism to assign incoming streams to different descriptor rings is TBD. The mechanism to determine the weights and relative priorities of each ring is also TBD.

Each of the four rings will be managed with an array of mblks corresponding to their location in the TX descriptor ring.

```
mblk_t  *ce_tmblkp[CE_TMDMAX];
```

Packets are passed to the ce driver via ce_wput(), where they are processed according to the type of mblk. Data packets are passed to ce_start(), where they are mapped for DVMA and placed in the transmit descriptor ring. The Cassini+ Transmit descriptor is programmed as follows:

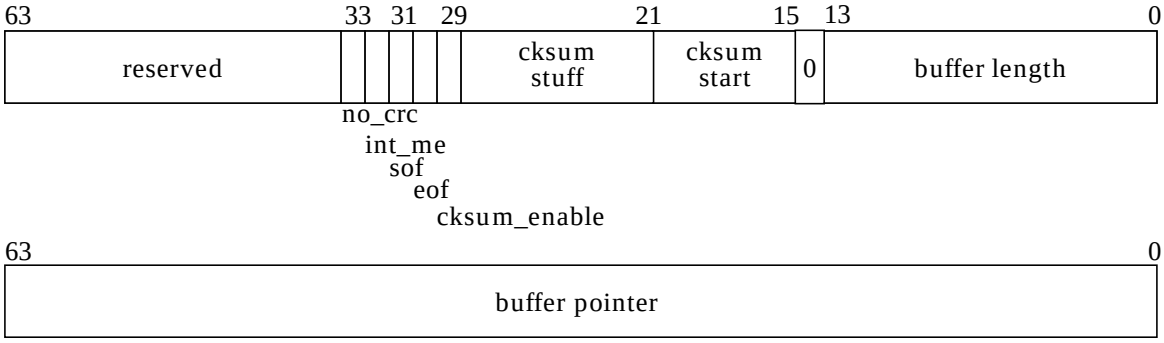


Figure 3-2 Cassini+ TX descriptor format

buffer pointer[63:0] : Pointer to the first data byte of the transmit buffer.

buffer length[13:0] : Length of the buffer in bytes. Values from 0 to 9256 bytes are valid.

reserved[14] : This bit must be programmed to zero.

cksum start[20:15] : Number of bytes from the first byte of the packet to be skipped before checksum calculation begins. Value must be even (i.e. bit 15 must be zero). Only relevant for the first buffer of a packet with the cksum_enable bit set.

cksum stuff[28:21] : Byte offset within the packet of the first checksum byte. Value must be even (i.e. bit 21 must be zero) greater than or equal to eight. Only relevant for the first buffer of a packet with the cksum_enable bit set.

cksum_enable[29] : When set to ‘1’ for the first descriptor of a packet, checksum is stuffed for the given packet.

end of frame[30] : When set to ‘1’ indicates the last descriptor of a packet.

start of frame[31]: When set to ‘1’ indicates the first descriptor of a packet. In normal TCP/IP operation roughly 75% of packets consist of a single descriptor, so SOF and EOF will both be 1. Another 20% of the packets have two descriptors, and the remaining 5% consist of three descriptors. Packets with more than three descriptors are possible but are extremely rare in the Solaris TCP/IP environment. With zero-copy TCP enabled, 10% of transmit packets consist of a single descriptor, 65% consist of two descriptors and 25% consist of three.

int_me[32] : When set to ‘1’ in the first descriptor of a frame, the TX_INT_ME interrupt will be set upon loading the entire frame into the TX FIFO. Typically the driver will set this bit every N frames but will leave the TX_INT_ME bit masked in the interrupt register until the descriptor ring fills and mblks must be queued. Once this happens the TX_INT_ME bit will be enabled and the interrupts used to quickly unload the descriptors.

no_crc[33] : When set to ‘1’ in the first descriptor of a frame, the CRC will not be inserted into the outgoing frame. This bit is for debugging and fault insertion purposes, and will not be used in normal operation.

reserved[63:34] : The driver will program these bits to zero.

3.2.3 Transmit Completion Handling

Cassini+ provides a “virtual completion register” by writing the values of the current completion of each of the four TX descriptor rings to the Completion Address location. The device driver will use these values as offsets into the ce_tmblkp array to determine which mblks can be freed.

In normal operation the ce driver will run with all three TX interrupt bits masked off. TX completion will be checked at the bottom of ce_start, and mblks freed as necessary. The int_me bit will be set every N packets.

When a descriptor ring fills the driver must resort to queueing mblks in the STREAMS queue. When this happens it will enable TX_INT_ME in the mask register, and use the interrupts to quickly drain the descriptor ring so that queueing can be avoided.

3.2.4 Transmit Zero-copy TCP Implementation

Some discussion of the implementation of zero-copy TCP in Solaris will be necessary to understand the workings of the Cassini+ device driver software. Unix applications send packets on the network using a system call such as write(), which gives the kernel the address of a buffer in the user’s address space to send. When the write() call returns the user is free to modify the buffer in their address space, which will not disrupt the processing of the packet already sent. Additionally the contents of the user’s buffer cannot be modified by the kernel during the processing of the write() call.

The semantics of write() assume that the kernel will copy the user’s buffer into a new kernel buffer before returning from the write(), and that this new kernel buffer will be used to actually send the packet to the hardware for transmission. As a performance optimization the kernel computes the TCP checksum while the data is being copied from the user buffer to the kernel, so that only one pass over the data is made.

The goal of zero-copy TCP is to get rid of that pass over the data by allowing the hardware to DMA directly from the user’s buffer. The most straightforward implementation would be to not return from the write() call until all processing had been completed and the user’s buffer had been fully DMAd by the hardware device. Unfortunately this leads to poor application performance because they cannot overlap any of their own processing with the serialization of packets.

The solution chosen in Solaris is to clear the write-permission bit on the user’s page while processing the write() call. That way if the user attempts to modify the buffer before the hardware has finished its DMA, they will trap into the kernel due to insufficient permissions. This is commonly called COW, for Copy On Write, and is used frequently in the kernel to avoid copying data unless

absolutely necessary. The app can be resumed as soon as the write permission has been removed, and will be able to perform other processing or work with other buffers while the network hardware is DMAing the one marked COW. When the hardware finishes DMA the kernel will restore the write permission to the user's page, and the app is then free to modify that page.

This scheme relies on the application to not attempt to modify their buffers for some amount of time after their write() call returns. There is no way for the app to know when it is safe to modify the page. However in practice we find that cycling between 3 or more buffers allows an app to avoid the COW fault in the vast majority of cases. The kernel keeps track of how often an app takes the COW fault, and will disable the use of zero-copy TCP for that application if too many faults are incurred.

Thus the user's data can be DMAd directly from the user's buffer, without a copy in the kernel. Any protocol headers such as the TCP and IP headers must be stored in another buffer, and the hardware is responsible for generating the outgoing TCP checksum.

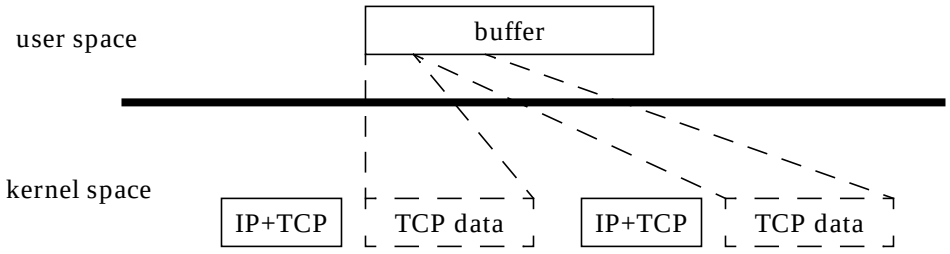


Figure 3-3 Transmit zero-copy example

3.3 Driver Receive Path

The Cassini+ driver departs from previous ethernet drivers more radically in its receive handling.

3.3.1 Receive Zero-copy TCP Implementation

Unix applications receive packets from the network using the read() system call, which gives the kernel the address of a buffer in the user's address space in which to place the incoming packet(s). The user can supply a buffer large enough to hold several packets, and kernel will copy the data from each packet one after the other.

Zero-copy TCP on receive works if the user supplies a buffer which is page-aligned and is a multiple of the page size. The device driver must provide the received packet in a buffer where the start of the TCP data is page aligned, and

there must be at least one full page of data in the buffer. If all these things are true the kernel can switch the MMU mappings between the page supplied by the user and the kernel buffer.

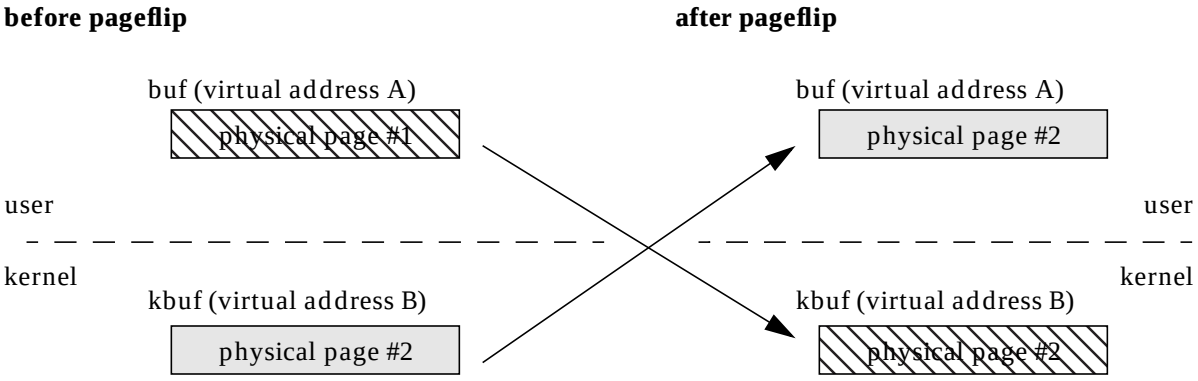


Figure 3-4 Receive pageflip example

Cassini+ endeavors to reassemble received packets from a TCP connection into a single buffer so that they may be pageflipped.

3.3.2 Receive Data Structures

Cassini+ utilizes a single buffer ring with each buffer expected to be page sized and page aligned. The hardware will place a number of packets into each page sized buffer, either by reassembling a flow or by dividing up the buffer into multiple packet buffers. Additionally the hardware might hold onto the buffers for some time while waiting for additional packets to arrive on a flow, meaning that buffers may be returned in a different order than they were placed on the descriptor ring.

Cassini+ maintains a buffer descriptor ring and a separate completion ring. The buffers are tracked using index handles which are placed in the descriptor ring and returned in the completion ring. The driver uses the index values to track which buffer is being returned, and so does not rely on buffers being returned in the same order they were placed on the ring.

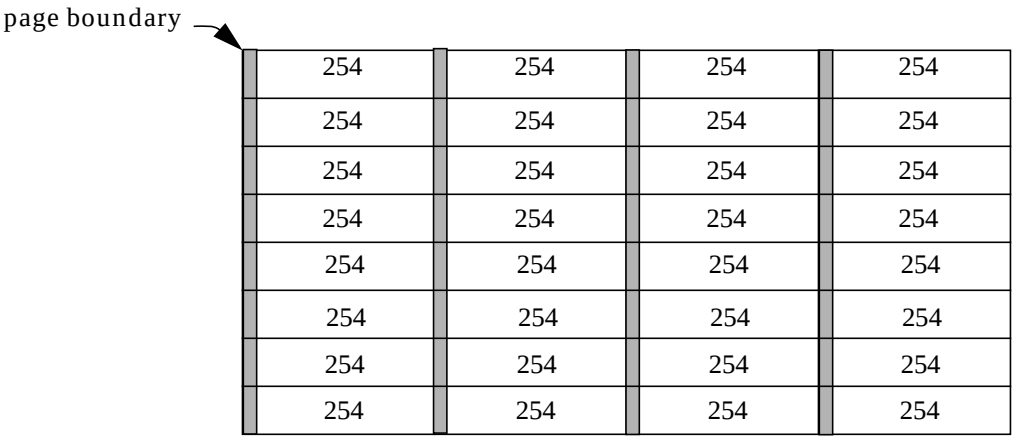
There is a table with information about the buffers which is indexed using the values passed to the hardware.

3.3.2.1 Multiple packets in a buffer

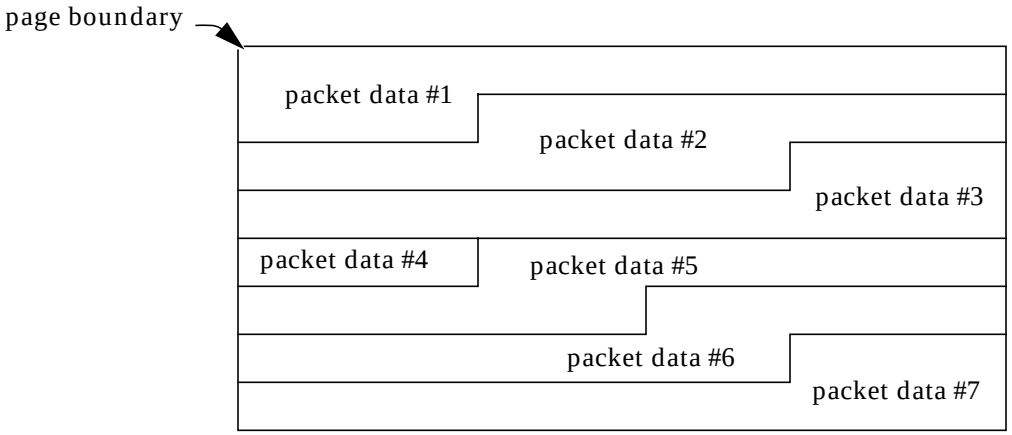
The RX buffer ring contains buffers which are page-sized, expected to be 8 KBytes on platforms supported by Cassini+ . The Cassini+ hardware will place multiple ethernet frames into a single buffer in one of three ways.

The first mechanism is used for small packets which do not fit into a flow, and for the protocol headers split off from packets which are re-assembled. A single 8 Kbyte page is broken up into 32 buffers of 256 bytes each. The hardware inserts 2 bytes of padding at the beginning of each packet (to get 32 bit

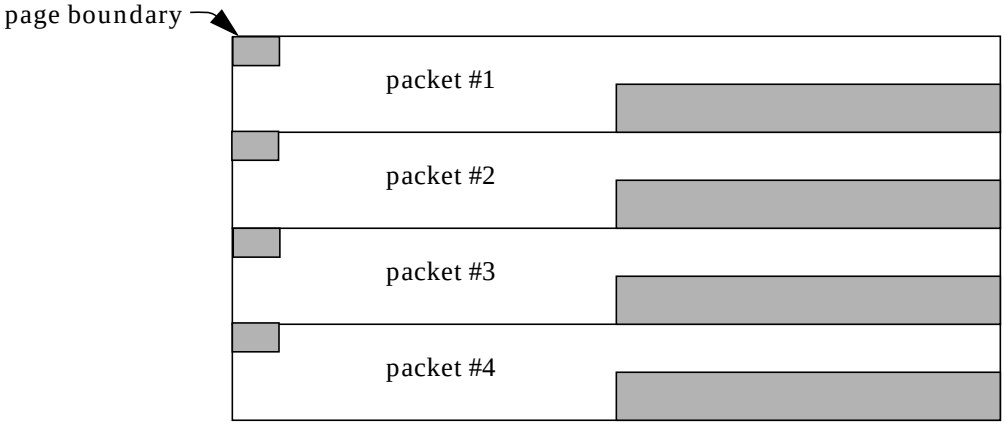
alignment at the start of the IP header), leaving room for 254 bytes of packet data. It is expected that software will allocate a new mblk and copy each of these small frames.



The second mechanism is used for flow reassembly. Each packet received by the Cassini+ hardware will be broken into two pieces: the LLC, IP, and TCP headers will be placed in a small buffer (as the preceding paragraph) while the packet data will be packed together to form a contiguous page. In this case there could be any number of packets placed into a single physical page, depending on the exact size of the packets as they arrive.



The third mechanism is used to receive packets which are larger than 256 bytes but which are not candidates for reassembly. Packets are placed in buffers spaced equally throughout the page. The hardware will skip a cache line at the beginning of each packet to leave room at the beginning of the mblks sent up to IP (important for routing performance).



3.3.2.2 *Mutex locking for rxbufinfo table*

At any given instant in time, a receive buffer can be in one of three places: on the descriptor ring owned by the hardware, on the completion ring owned by the software, or in the middle of processing by ce_read(). Transitioning a buffer from one of these three places to another is already protected by a mutex lock: the descriptor ring has a lock protecting it, and the interrupt handler has intrlock. Thus, access to the rxbufinfo table can be controlled by ownership of the index, only the thread which is currently processing that index is allowed to access that row in the rxbufinfo table. No additional locking is required for the table.

3.3.3 *Batching*

The Cassini+ hardware parses incoming TCP/IP packets to classify them into flows. It can have up to 64 flows in its database. When a packet is handed over to the software, the hardware will look ahead in its internal fifos to determine if another packet from the same flow has arrived and is awaiting DMA. If so, the hardware will indicate that the flow has additional packets available, encouraging the driver to hold off processing this packet until the subsequent packets have arrived. In this way the CPU cache can be more effectively used, and protocol techniques such as header prediction will be more effective.

To support batching the driver must maintain a data structure for each of the 64 flows, and may store mblks temporarily in this structure if indicated to do so by the hardware.

3.3.4 *Multi CPU receive*

By parsing the packet headers to identify flows, the Cassini+ hardware can identify the source and destination IP addresses, and the TCP port numbers. These fields uniquely identify a connection. The hardware will hash these

fields and take the modulo over the number of CPUs in the system. This provides an index which can be used to distribute processing load across multiple CPUs. Because packets from a particular <IP src, IP dst, TCP src, TCP dst> tuple will always hash to the same index, packets from a connection will always be sent to the same CPU and will thus stay in order.

There are several schemes which can be used to distribute receive processing amongst multiple CPUs. In the sun4u architecture a mondo-vector can be sent to a remote CPU to trigger a trap, which can then dispatch a softinterrupt on that CPU. Experiments show good results using this technique (a 4 CPU machine could receive 3X the data rate of a single CPU machine).

Threads can also be used to load balance amongst different CPUs. Although thread dispatch is considerably more expensive than cross-traps, the system may benefit from making network processing more like normal user processing.

3.3.5 *Receive Descriptor Ring*

Free buffers are handed to Cassini+ on a descriptor ring which contains page-sized buffers. The descriptor format is the same for each.

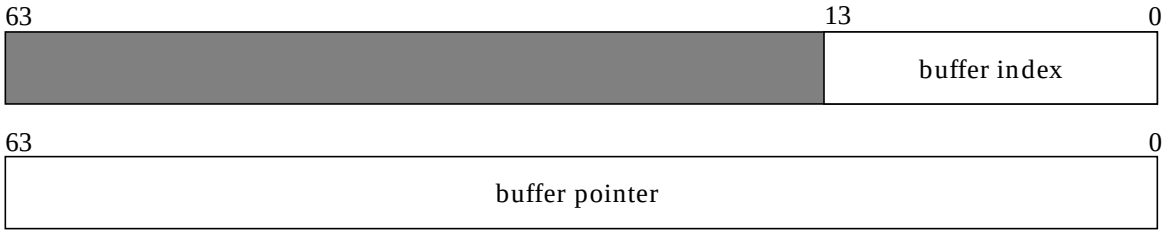


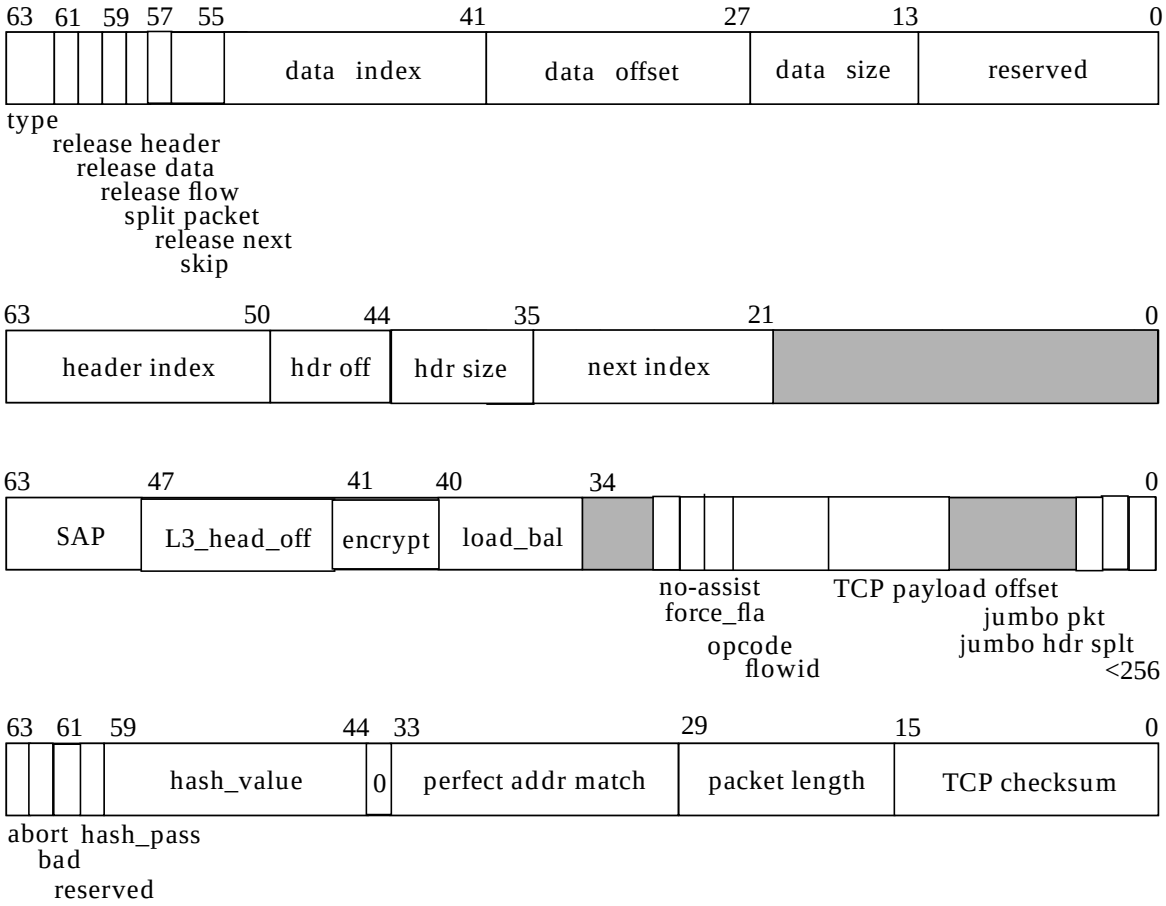
Figure 3-5 Receive Descriptor

buffer index[13:0] : The index within the rxbufinfo table for this buffer. The hardware does not interpret the index value in any way, it merely stores it and returns it in the completion ring.

buffer pointer[63:0] : The DVMA address for this buffer.

3.3.6 Receive Completion Ring

As Cassini+ receives packets it returns them to the software on the completion ring. Since multiple packets can be placed within a single buffer during reassembly there will likely be far more completion ring activity than descriptor ring. Also since packets for a particular flow can arrive in any order, the buffers may be returned to software in any order.



Word 1:

type[63:62] : Indicates ownership of this descriptor.

00 - owned by hardware

01 - release of stale flow buffer

10 - release of non-reassembled packet

11 - release of flow packet

To the driver, 01, 10, 11 are treated identically and indicate a packet arrival.

release_header[61] : Indicates this is the last packet to fit into a header buffer. The driver will recycle the header buffer back to the descriptor ring when this bit is set.

release_data[60]: Indicates this is the last buffer to fit into a flow reassembly buffer. The driver uses this bit to determine if it is safe to free the page.

release_flow[59]: This bit is only used if batching is enabled in the driver. If un-set, i.e., 0, the hardware has determined that there are additional packets from the same flow present in its control FIFO. The driver will queue this packet in a data structure until the subsequent packets from the same flow arrive. If 1, the driver will send up all packets on this flow. Jumbo packets and non-reassembly packets will always have this bit set.

split_packet[58]: Indicates this packet spanned two reassembly buffers, and that the next_index field is valid. The driver will process a split buffer packet when this bit is set.

release_next[57]: Indicates this is the last buffer to be placed in the "next_index" reassembly buffer. The driver uses this bit to determine if it is safe to free the page.

skip[56:55]: The number of completion entries to skip after the current entry. Contents of descriptors marked for skip should be ignored. The driver will use this field to find the next live completion entry after the current entry.

data_index[54:41] : The index of the data buffer being returned (which was set in the RX descriptor ring). The driver uses this to index into its rxbufinfo array.

data_offset[40:27] : The offset within the data buffer of the start of this packet. The driver uses this to locate the packet.

data_size[26:13] : The length in bytes of the data portion of the packet. A length of zero indicates there is not data buffer.

reserved[12:0]

Word 2:

header_index[63:50] : Index of the header buffer. The driver uses this to index into its rxbufinfo array.

header_offset[49:44] : Offset in 256 byte units into the current header buffer. The driver uses this to locate the beginning of the header.

header_size[43:35] : Length of the header in bytes. A length of zero indicates there is no header buffer for this packet.

next_index[34:21] : The index of the buffer containing the rest of the packet, in the split packet case. The offset is automatically zero for the second buffer.

reserved[20:0]

Word 3:

SAP[63:48] : The ethertype. The driver will use this to determine which stream to send the packet to.

L3_head_off[47:42] : Offset in double-bytes of the first byte of the L3 header. The driver will use this when stripping off ethernet and LLC headers before sending the packet upstream.

Encrypted_pkt[41] : An IPsec's AH or ESP encrypted packet is detected via its IPv4 or IPv6's base header.

load_bal[40:35] : A suggested load balancing key. The driver may use this to dispatch interrupts to other CPUs in the system.

reserved[34:30]

no-assist[29] : Indicates Cassini+ was unable to parse the header of this packet. This bit is not used by the driver.

force_fla [28] : Indicates Cassini+ has done batching lookahead for opcode 2.

opcode[27:25]: Opcode, internal to Cassini+ . These bits are used to drive decisions on batching of packets in a flow.

flowid[24:19]: The flow id for this packet. These bits are used to drive decisions on batching of packets in a flow. The flowid is only valid for reassembly packets.

Checksum start offset[18:12] : The offset in double bytes to the first byte of data included in the checksum.

reserved[11:3]

Jumbo_hdr_split_enb[2] : If set, jumbo packet headers should be split off into a header buffer.

Jumbo_packet[1] : This packet's payload exceeds 1522 bytes.

small_pkt[0] : This bit is set to 1 if the packet is less than or equal to (256 bytes - swivel_offset) in length.

Word 4:

length mismatch[63] : Packet length field from MAC did not match actual packet length. Driver will discard the packet.

bad[62] : Packet has a CRC error, is a runt, etc. Driver will discard the packet. This would only be set when disable_discard_on_err bit is set in the MAC.

reserved [61] : reads zero.

hash_pass[60] : The packet hashed into a multicast bucket which is turned on in the hardware. The driver will optionally do a second level of filtering, and pass the packet up to software.

hash_value[59:44] : The mutlicast address hash value. The driver will use this to perform a second level of filtering.

zero[43] : The hardware always writes this bit as zero. This bit serves as an OWN bit and is set by the driver when setting up completion ring entries prior to hardware utilization. This bit is written to host memory during

the final data phase transfering the 32 byte completion ring entry over the PCI bus so when the driver observes this bit clear it can safely conclude the entry has been completely updated by hardware.

perfect_addr_match[33:30]: Packet matched one of the multicast address registers. If any of these bits are set the packet will be accepted by the driver.

packet_length[29:16] : Length in bytes of the entire packet. This field will not be used by the driver.

TCP_cksum[15:0] : The calculated checksum over this packet. The driver will place this value into the b_ick_cksum field of the mblk before sending it up. For reassembly packets or jumbo packets with header split, the checksum is done from the checksum_offset to the end of the TCP payload. For non-reassembly packets, checksum is done from the checksum_offset to the end of the packet. Checksum is given in big endian format.

3.3.7 Pseudocode for driver receive path

3.3.7.1 Atomic counter utility functions

The Cassini+ driver must maintain complex data structures relating to receive memory management, due to a two level allocation scheme. The driver allocates a page-sized buffer which is carved up into multiple packets by the Cassini+ reassembly hardware. The driver cannot free the page of memory until all mblks placed within it have been released by the upper layer protocols.

Therefore the driver must maintain a reference count for each page of memory, counting the number of mblks which have been placed on that page. Since mblk allocation and release are asynchronous the driver must maintain consistency of this reference counter amongst multiple threads of execution.

To avoid a preponderance of mutex locks the Cassini+ driver will rely on atomic instructions implemented in the Sparc v9 architecture to maintain reference counters for each page of memory. The utility routines to do this are shown below. Note to implementors: these routines should be placed in a separate .s file and compiled with “as -P -D_ASM -D_KERNEL -xarch=v8plus”.

Note – Routines similar to these are included in Solaris 2.6 and later in all architectures, as atomic_add_32_nv() (call with -1 to subtract). The Solaris routines should be used with 2.6 and later rather than the implementations shown here.

```
      .file "atm_v9.s"
#include <sys/asm_linkage.h>

      ! register usage (no window spill is done):
      ! %o0 : input, pointer to the integer count
      ! %o1 : input, amount to decrement from *count
      ! %g1 : scratch, original value of *count
      ! %g2 : scratch, new value of *count
      ENTRY_NP(adecr)
1:
      ld       [%o0],%g1          ! move *count into register %g1
```

```
        sub     %g1,%o1,%g2         ! %g2 = %g1 - %o1
        cas     [%o0],%g1,%g2! if [%o0] == %g1, swap %g2
                                    ! and [%o0] atomically.
        cmp     %g2,%g1             ! if the swap was not successful...
        bne     1b                  ! ... go back and do it again
        nop                         ! branch delay slot, nothing to do
        retl
        sub     %g1,%o1,%o0         ! return new val (branch delay)
        SET_SIZE(adecr)


        ! register usage (no window spill is done):
        ! %o0 : input, pointer to the integer count
        ! %o1 : input, amount to decrement from *count
        ! %g1 : scratch, original value of *count
        ! %g2 : scratch, new value of *count
        ENTRY_NP(aincr)
2:
        ld      [%o0],%g1           ! move *count into register %g1
        add     %g1,%o1,%g2         ! %g2 = %g1 - %o1
        cas     [%o0],%g1,%g2! if [%o0] == %g1, swap %g2
                                    ! and [%o0] atomically.
        cmp     %g2,%g1             ! if the swap was not successful...
        bne     2b                  ! ... go back and do it again
        nop                         ! branch delay slot, nothing to do
        retl
        add     %g1,%o1,%o0         ! return new val (branch delay)
        SET_SIZE(aincr)
```

3.3.7.2 *receive routine*

```
int
ce_read(ce_t *cep, ce_rxc_t *crxp) {
      int        didx;
      mblk_t     *mp = NULL;
      uint64_tdword0;
      rxflow_t*fp;

      ...

      dword0 = crxp->dword0;

      /*
       * if there is a header buffer, allocb an mblk and copy the
       * header into it.
       */
      if (dword0.header_size) {
            int   hidx = dword0.header_index;
            caddr_tp;

            /* use the buffer index to locate the base of the buffer */
            p = cep->rbufs[hidx].buf + (dword0.header_offset * 256);

            mp = allocb(dword0.header_size + 64, 0);
            mp->b_rptr += 64;
            bcopy (p, mp->b_rptr, dword0.header_size);

      }

      if (dword0.release_header) {
                  ... place the page back on desc ring ...
      }


      /*
       * if there is a data buffer, wrap an mblk around it.
       */
      if (dword0.data_size) {
            caddr_t    p;
            ce_esbrtn_t*esbp;

            didx = dword0.data_index;

            esbp = cep->rxbufs[didx].esbrtnp;
```

```
      p = cep->rxbufs[didx].bufp;
      nmp = desballoc(p, dword0.data_size, 0, esbp);

      if (mp == NULL)
            mp = nmp;
      else
            mp->b_cont = nmp;

      if (dword0.release_data) {
            ddi_dma_unbind_handle (handle);
            cep->allocneeded++;
      } else {
            aincr(&esbp.refcnt);
      }
}

if (dword0.next_size) {
      mblk_t *nnmp;
      caddr_t    p;
      ce_esbrtn_t*esbp;

      nidx = dword0.next_index;

      esbp = cep->rxbufs[didx].esbrtnp;
      p = cep->rxbufs[didx].bufp;
      nmp = desballoc(p, dword0.data_size, 0, esbp);

      nnmp = mp;
      while (nnmp->b_cont != NULL)
            nnmp = nnmp->b_cont;

      nnmp->b_cont = nmp;

      if (dword0.release_next) {
            ddi_dma_unbind_handle (handle);
            cep->allocneeded++;
      } else {
            aincr(&esbp.refcnt);
      }
}

/* Allocate replacement buffers */
while (cep->allocneeded > 0) {
      if (ce_allocbuf())
            break;
      cep->allocneeded--;
}

fp = &cep->rxflow[dword1.flowid];

/* Do packet batching */
if (fp->last == NULL) {
      fp->first = fp->last = mp;
} else {
      fp->last->b_next = mp;
      fp->last = mp;
}

if (dword0.release_flow) {
      /*
       * Note: the putnext() will be done from a softint,
       * thread, or rsrv routine and not in the hw
       * interrupt. The exact choice is not germane here.
       */
      mblk_t *mp, nmp;
      mp = fp->first;
      nmp = mp->b_next;
      while (mp != NULL) {
            mp->b_next = NULL;
            queue_packet (wq, mp, word4.load_bal);
            mp = nmp;
            nmp = nmp->b_next;
```

```
            }
      }
}


static int
ce_allocbuf(ce_t *cep, int index) {
      ce_rxbufinfo_t*rxbp = &(cep->rxbufs[index]);

      rxbp->buf = allocate a page of memory (ddi_page_alloc?)

      rxbp->esbrtnp = kmem_alloc(sizeof(ce_esbrtn_t));

      rxbp->esbrtnp->buf = rxbp->buf;
      rxbp->esbrtnp->refcnt = 1;
      rxbp->esbrtnp->frtn.free_func = ce_reclaimbuf;
      rxbp->esbrtnp->frtn.free_arg = rxbp->esbrtnp;

      if 4 buffers have been allocated, post them to the hardware.

      if any allocations failed return 1, otherwise 0
}


static int
ce_reclaimbuf(ce_esbrtn_t *esbp) {
      if (adecr(&esbp->refcnt) == 0) {
            page_free esbp->buf page.
      }
}
```

PCI Interface and Core Arbitration 4

4.1 PCI Bus Interface (BIM) block

The Bus Interface Module (BIM) interfaces the Cassini+ core logic to the PCI bus interface of Cassini+ . The core and BIM are autonomous units in that there is a minimum amount of logical visibility from one side into the other and the two sides operate from independent clocks. The core runs at 125 MHz and the bus interface module runs at the PCI clock rate which can be set between 1 MHz and 66.6 MHz. Naturally, necessary portions of the bus interface module will be clocked from the core clock as needed to couple the two units.

4.1.1 Basic Features

Cassini+ 's PCI interface includes the following:

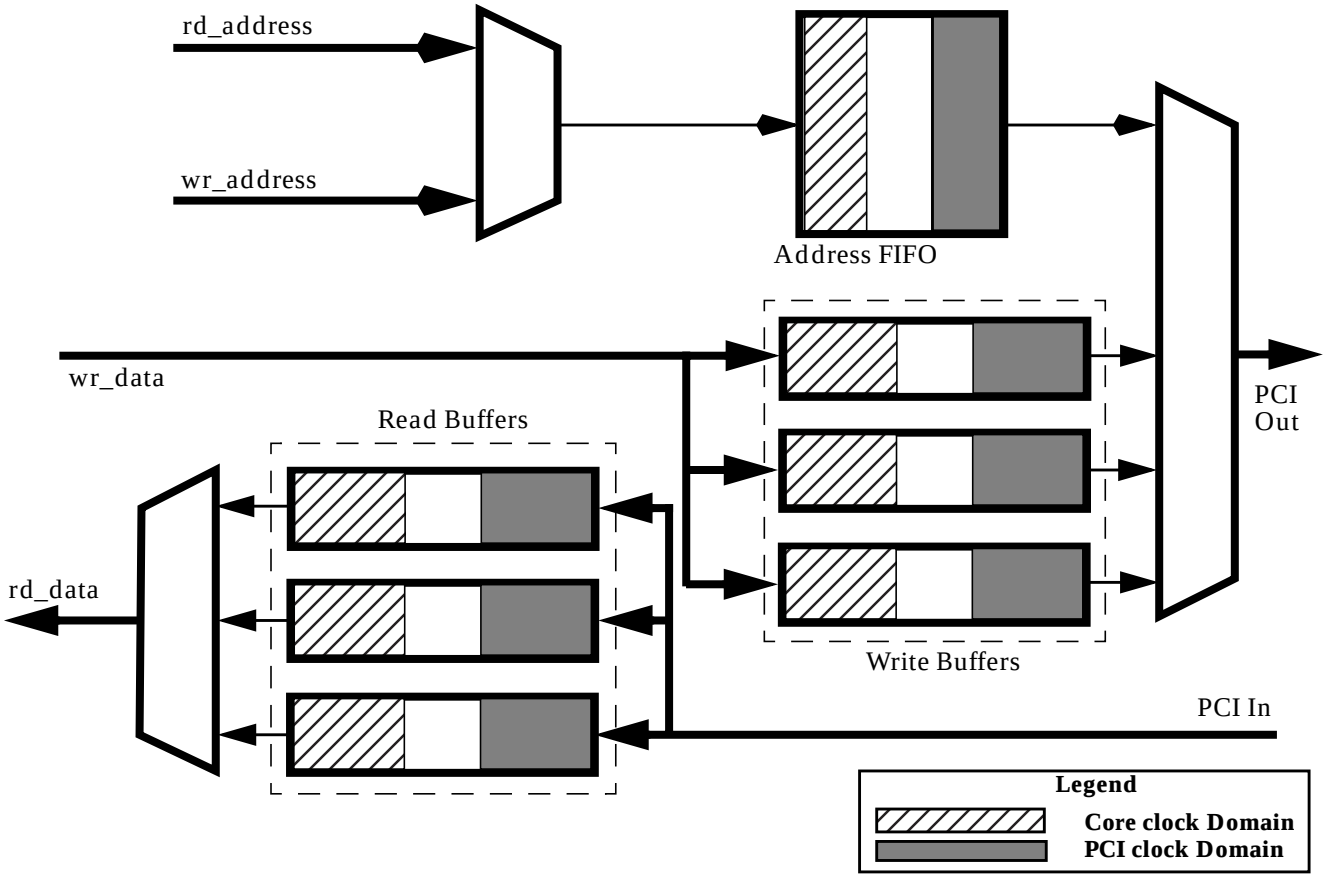
- Compliant with PCI 2.2 Local Bus Specification
- 66.6 MHz operation with 33 MHz capability
- Capable of sub 33 MHz operation (with corresponding performance degradation) for power savings
- Universal I/O supporting 3.3V and 5V signal switching
- 64 bit master data path with 32 bit system support
- Supports 64 bit master addressing and DMA operation in addition to DVMA
- 32 bit slave data path
- Supports byte wide prom interface
- New signal interface protocol between core and bus interface to support asynchronous operation between independently clocked units
- CompactPCI
- Hot Swap CompactPCI(PICMG 2.1 R1.0)

4.1.2 Endianess

As a PCI device, Cassini+ follows a little endian bus organization for all its blocks, spaces and transfer types.

4.1.3 Read and Write Buffer Synchronization Rings, Address FIFO

Pictorially, the synchronization buffers between core and PCI clock domains is shown below.



State machines in BIM within each clock domain track which buffers are used and which are empty in order to allow GPX signal protocol conformance and PCI bus master operations.

4.1.4 Performance Considerations

Cassini+ 's fundamental goal is to provide full duplex link rate operations. All portions of the design must be efficient to achieve this. The bus interface, among other things, will include the following features aimed at maintaining high efficiency:

- support for long bursts
- optimized bus transfers when target data width (either 32 or 64 bit) is known prior to beginning of burst
- BIM keeps bus request asserted when it knows a subsequent bus operation is pending
- spacing between consecutive bus transactions is kept at a minimum

4.1.5 PCI Bus Logic Detail

The BIM functions as the interface between the PCI bus and the backend core logic. The PCI core works at the PCI clock and provides interface for the slave and the master cycles on PCI side. This portion of the BIM block is totally synchronous with the PCI clock. Data transfer in the read and write direction is stored or taken from the synchronization buffers provided in the BIM block. This section describes the various aspects of the PCI interface in the slave and master modes.

4.1.5.1 PCI Slave Interface

The slave interface for Cassini+ is a 32 bit wide interface from the PCI point of view. The slave accesses fall in one of the following categories. They are, the configuration access, Cassini+ I/O register access and the expansion ROM access. The details of this are given in 2.4.6. Following are the salient points pertaining to the slave interface.

- Cassini+ slave interface does not support burst access. The slave accesses are terminated by asserting TRDY#, signaling a termination with data transfer to the external master. In the event of the external master intending a burst access, Cassini+ disconnects the cycle with data.
- Supports linear increment addressing mode. The discontiguous or misaligned byte enables are not supported by the Cassini+ slave interface. Under these situations, Cassini+ issues a target abort.
- The slave interface does not support PCI commands like interrupt acknowledge (0000), special (0001), I/O read (0010), and I/O write (0011). The MRM and MRL commands are handled as a memory read command. Similarly the MWI is also treated as standard memory write command. The LOCK# signal is ignored during slave accesses.
- The Cassini+ slave interface adheres to the slave access latency as indicated in the PCI2.2 spec. This will allow a maximum of 16 PCI clocks as the initial latency and a maximum of 8 clocks for subsequent data phase. But the case of subsequent data phase does not arise due to the fact that Cassini+ slave interface does not support the burst access.
- Subsystem ID in the configuration space is provided according to the PCI2.2 specification.
- The configuration cycles support byte, half-word and word accesses.
- Provides support for the parity error handling based on the “Parity Error Response” bit in the configuration space. During address parity error the SERR# is asserted if the “Parity Error Response” bit is set and the cycle is ignored. In addition the “Detected Parity Error” bit in the configuration space is also set. During the data parity errors the slave interface responds by setting the “Detected Parity Error” bit and asserts the PERR#. This happens in the write cycles. The cycle is completed but the data will be ignored.

- As an enhanced feature for RAS it is planned to provide a definite number of retry termination for a slave write which has encountered the data parity error. As a result of this, the slave cycle will be retried. A status flag can also be set for indicating this error.

4.1.5.2 PCI Master Interface

The master interface of Cassini+ is responsible for the DMA data transfers between the PCI memory and the Tx and Rx blocks. This interface works at the PCI clock and exchanges data through the synchronization buffers of the BIM block. The following paragraphs explain the various aspects of the cycles initiated by Cassini+ when functioning as a master on the PCI bus.

Bus Commands and Bus Cycles

Cassini+ 's PCI master interface will not initiate the following commands through the C/BE# lines during the address phase. They are Interrupt acknowledge, Special cycle, I/O Read or Write and Configuration read or write. As a master, Cassini+ does not attempt for exclusive accesses and will not assert LOCK# at any time. Also the master will generate address using the linear increment mode.

Address and Datapath

The data path consists of two set of synchronization buffers, one in the read direction and the other in the write direction. Each set is a group of three buffers each of 64 bytes. These buffers are used to smooth the data flow between the PCI side and the core side which are operating at two different clocks. The core side is at 125 MHz and the PCI side can vary between 66 MHz and theoretically to zero.

The address path has a two level of buffering. The actual address for the DMA operation arrives from the Tx or Rx block. This address is stored in the address buffer. Along with the address, the size and type information is also stored in the address buffer. This address is sent out during the master transaction to be clocked into the PCI output registers. Using the size and the type information, and the cache line size, the master logic generates a suitable command output so as to use the PCI bus in an efficient manner.

Master Latency

The master interface adheres to the latency recommendations of the PCI2.2 spec. Master interface maintains a latency timer. During a long burst, if the latency timer expires and if the GNT# is deasserted by the arbiter, Cassini+ will give the bus ownership according to the PCI specification. Also the initial latency for master initiated accesses is kept within 8 PCI clocks.

Error Handling

- A master abort is signalled by the master when there is no target responding within the substractive decoding time by asserting the DEVSEL#. The Received Master Abort bit in the status register is also set.

- When a target abort situation happens on the bus, Cassini+ sets up the Received Target Abort bit in the status register. This cycle is not restarted.
- When a data parity error occurs during a master transfer, the Data Parity Error Detected bit and the Detected Parity Error bit in the configuration register is set. During a write operation, the target will be check for parity error and assert PERR# if it faces any error. During the read operations, Cassini+ 's master interface will check for the data parity and signal PERR# in case if there is an error. I both the cases, the master will generate an interrupt if the Parity Error Response bit in the command register is set.
- Similar to the situation in slave cycle, here also the feature for re-initiating the PCI cycle during parity error.

4.1.6 Slave PCI Accesses

The detailed register mappings for Cassini+ 's slave address spaces are described in Chapter 3 "Programmer's Model". The three slave access spaces are:

PCI Configuration Space

This is a 256 byte address space that follows the format of a single-function device. It is mostly a read / write space accessible as bytes, half-words or words, and must be accessible at all times. Values programmed into PCI Configuration space are not to be affected by software reset, and only return to their default values upon hardware reset.

PCI Configuration registers are implemented in the PCI block. Their decode starts at offset zero and is enabled by the IDSEL input (per slot addressing).

Cassini+ Register Space

Decodes for various blocks in the chip are generated by the PCI Interface block. The base address for this space is determined by the Base Address Register in PCI Configuration Space. Accesses to Cassini+ 's registers is limited to 32 bit word accesses, and is mapped as non-prefetchable PCI memory space.

Expansion ROM Space

The Expansion ROM Space is a read-only space typically accessed by the system during POST. While the decoding of this space is done by Cassini+ , the actual data is stored in the byte wide external PROM. Cassini+ performs byte stacking to support any combination of 8-bit, 16-bit and 32-bit accesses. The Expansion ROM Base Address Register in PCI Configuration space is used to define the base address using bits [31:20], while bits [19:11] are hardwired to zero to define a 1Mbyte space. The decoding of the Expansion ROM is enabled by bit 0 of this register and the Memory bit of the Command Register being both set. Physically Cassini+ supports up to 64kbyte PROMs, mapped at the beginning of this 1Mbyte space.

The Expansion ROM is also readable during run-time via the Register Space.

The delay introduced by byte stacking is handled in two possible ways:

- Accesses via the Expansion ROM space will complete after all required data bytes are read into Cassini+ . The PCI target initial latency rule does not apply in this case.

- For Accesses via Register space, if the byte stacking delay exceeds the target initial latency Cassini+ will terminate the cycle with a retry, and will continue prefetching up to 4 bytes in preparation for the cycle being retried by the initiator.

The PROM access timing is fixed at 10 EPCI clocks (5 PCI clocks).

PIO Interface Protocol

The PIO Strategy is described below.

1) All PIO accesses are 32 bits wide, with the following exceptions.

- Configuration space registers supports 8, 16 and 32 bit accesses.

- On board PROM supports 8, 16 and 32 bit accesses.

2) All blocks receive a dedicated chip select signal.

3) All blocks return a dedicated ready signal.

4) All blocks share the same 32 bit bus for PIO writes.

5) All blocks return PIO read data on a point to point, dedicated PIO read data bus.

6) All blocks return read data within a bounded number of clocks.

7) Write data for all blocks will be kept stable for a bounded number of clocks.

8) PIO reads from the PROM go the through the PROM controller block. The BIM block indicates the size of access to the PROM controller block. For 16/32 bit accesses, PROM bytes are packed to complete the access.

9) All PIO writes are registered in BIM.

10) The PIO reads that cannot be completed in 16 clock cycles will be issued a retry. Except PROM accesses, all other PIO read accesses are completed within 16 clock cycles.

4.1.7 PCI DMA Functions

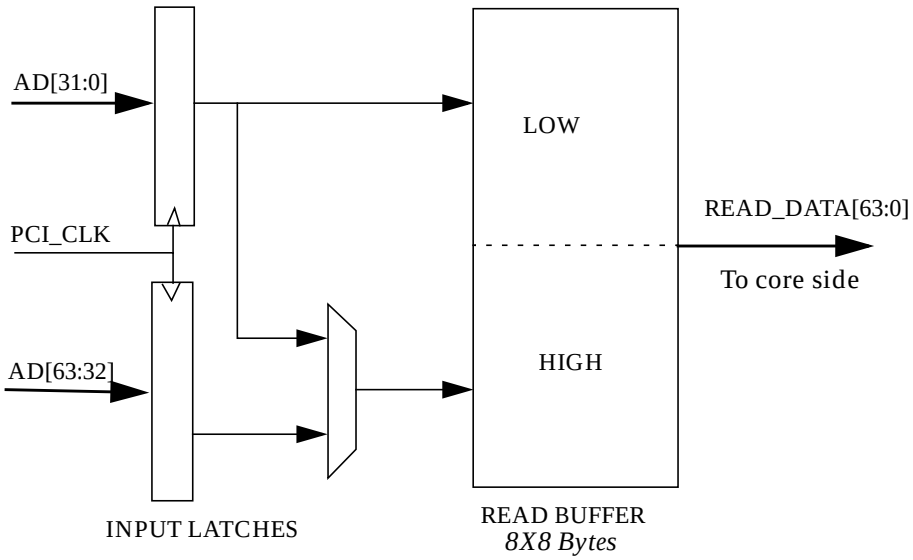
Data and descriptor movement to and from host queues is done by Cassini+ as a PCI DMA bus master. As a Gigabit full duplex device, it is important to optimize the bandwidth of Cassini+ 's memory access functions, with the characteristics of Sun's PCI platforms in mind.

As for the actual DMA behavior, when allowed by the DMA target and the arbiter, Cassini+ is capable of bursting entire frame lengths to maximize efficiency on the bus. This capability is achieved by large FIFOs, and by having enough internal bandwidth between the PCI and the FIFOs to sustain such long bursts.

The following section explains the various aspects of PCI transaction during DMA cycles.

4.1.7.1 DMA Burst (read) Cycles

The following figure shows the data flow during the burst read accesses as a result of the DMA. This diagram shows one of the three buffers available in the read direction.



The data from PCI bus is latched in using the PCI clock. From the PCI clock domain the data is written into the 64 bit wide buffer. Once the buffer becomes full and if the second buffer is empty, the read continues with filling of another 64 bytes. This continues for the third buffer also. Thus the three data buffers function as a FIFO to match the different rate on the two sides of the buffers. When the PCI is capable of handling 64 bit wide data, the writes into the read buffer are done with the same width (64 bits). Hence there will be eight data transfers to get one buffer full. If the bus happens to be 32 bits wide, then the low word and high word portion of the read buffer are written alternatively so that the buffer will become full after 16 data transfers on the PCI bus. This enables the core side bus to provide the data in a 64 bit bus.

Memory Read Command Usage

The PCI memory command usage during the DMA read operation given here. The Cassini+ PCI master read interface issues a Memory Read Line (MRL) when there request from the core to perform exactly one cacheline size. The BIM can decide on the proper command based on the size information requested from core and the system cacheline size that has been programmed during the initialization cycle. For cases when the core has requested transfers of multiple cacheline size, then Cassini+ will issue a Memory Read Multiple (MRM) command, so that the target can prefetch a set of data beforehand for

the PCI master to read. When the access size is less than a cacheline size, Cassini+ will initiate a Memory Read (MR) command. (TBD). Any time the MRL or the MRM commands are disconnected, Cassini+ has to restart the cycle from the location where it was disconnected by the target. During these cycles PCI master interface issues a MR command to read the reaming words of the burst.

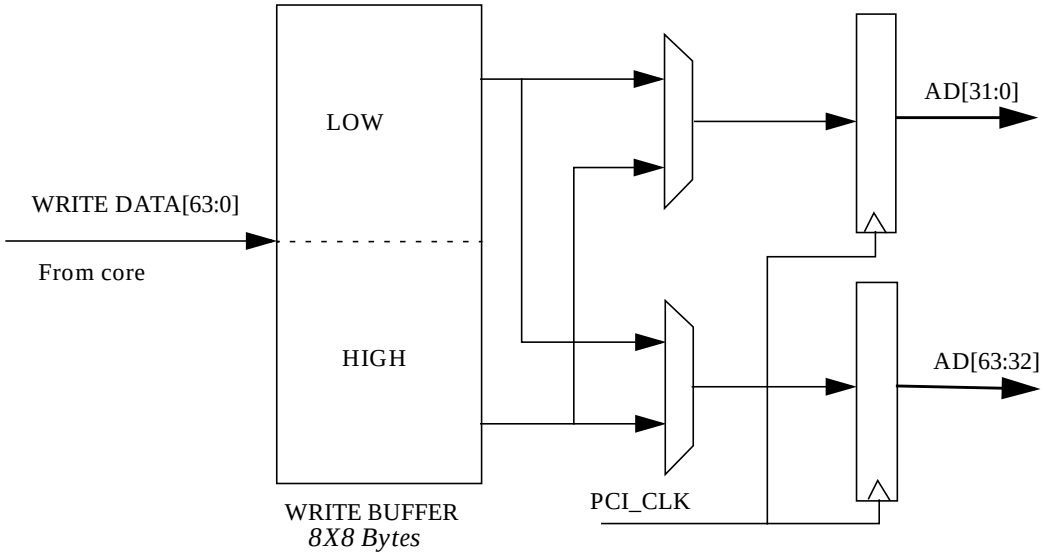
Read Burst Behavior

The master read interface adheres to the following points during the DMA

- Master issues a single burst with the minimum data phases which satisfies the core's read request. In the event of getting disconnected from the target, this interface generates reads from where it was stopped.
- Master will do 64 bit transfer whenever possible. If the target is only 32 bit capable or if the 64 bit transfer is disabled, the master will initiate a 32 bit transfer.
- Cassini+ supports a cacheline size of 16, 32, 64 and 128. When the cacheline register is programmed with 0 or with any other unsupported values, the master interface defaults to a cacheline size of 64.

4.1.7.2 DMA Burst (write) Cycles

The data flow during the burst write transaction is depicted in the figure below. The figure shows one of the three buffers.



Write data from the core is transferred into the write buffer when the buffer is declared free by the BIM. The core provides the address, size and the type information at the time of giving the request for the data transfer. Once the buffer is full or the requested number of bytes have been transferred, the Cassini+ PCI master initiates a write transaction on the PCI side. Depending on the bus width of the target, the data from write buffer is channelized into

the PCI data pins in a proper manner. At the end of the transfer, the core is indicated about the completion of the transfer by the xfr_done signal. In the case of any disconnects, the cycle is rescheduled from the point of disconnect.

Memory Write Command Usage

There are two possible write commands in PCI. The Memory Write (MW) and the Memory Write Invalidate (MWI). The MWI command helps the target to identify that Cassini+ is willing to transfer one complete cacheline so that the target need not perform a cache consistency operation. Cassini+ issues a MWI command when the core has requested at least one full cacheline to be transferred. Under other conditions, Cassini+ issues a simple MW command.

Write Burst Behavior

During the DMA write, the master interface adheres to the following behavior

- The master issues write commands in the burst mode if it has to do an integer number of cacheline size transfers. For accesses starting on any address the burst will start with that address and continue till the end of the cacheline.
- Master will do 64 bit transfer whenever possible. If the target is only 32 bit capable or if the 64 bit transfer is disabled, the master will initiate a 32 bit transfer.
- Cassini+ supports a cacheline size of 16, 32, 64 and 128. When the cacheline register is programmed with 0 or with any other unsupported values, the master interface defaults to a cacheline size of 32.

4.1.7.3 DMA address generation

a. Capable of generating 64 bit addresses for descriptors and data

b. Only the lowest 50 address bits are manipulated for pointer generation, the higher 14 bits are taken as is from the appropriate register / descriptor

c. Dual Address Cycles are used when the upper 32 address bits are non-zero

d. Byte resolution addresses issued for reads and writes

4.2 Core Arbitration Block

There are multiple DMA sources which need to compete for the GPX bus interface conduit to host memory. The host CPU also needs to access registers, memory, and other slave device elements within the Cassini+ core. DMA sources within Cassini+ must adhere to a policy limiting the extent which one source excludes another. The core arbitration block handles DMA requests from the TX and RX units and is the concentration point for slave access signals.

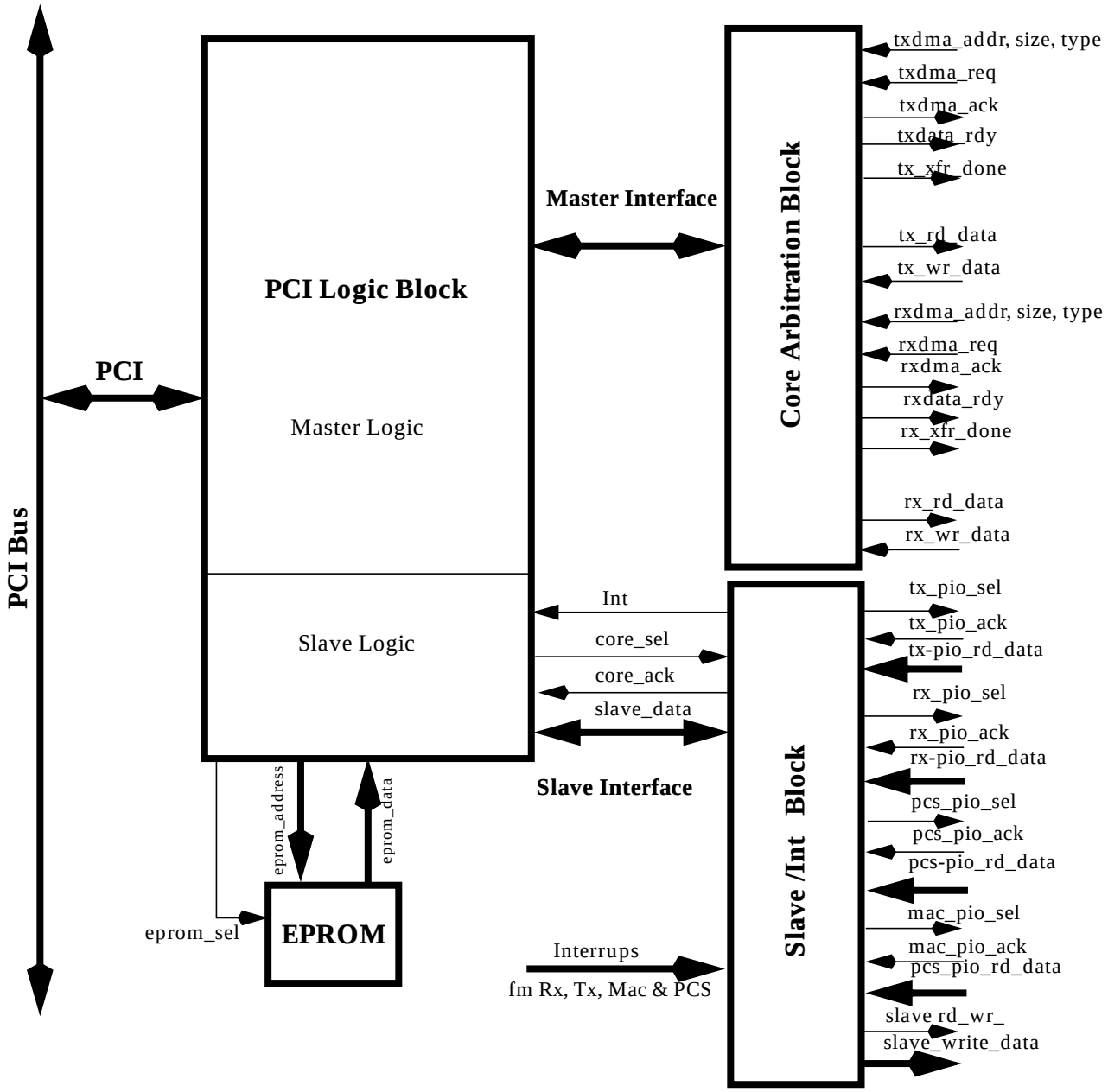
The presence of the TX and RX FIFO allows systems that support long DMA bursts on PCI to increase bus transfer efficiency by minimizing the need to switch between TX and RX DMA channels too frequently.

The “TX_DMA_Limit” and “RX_DMA_Limit” values programmed in the Configuration Register are meant to address the side effects of long burst DMA, when both channels are active (increased latency, bandwidth mismatch due to packet size mismatch).

Programming different DMA_LIMIT values for the TX and RX channels also constitutes a mechanism for controlling the relative ratio of the bus that each channel will use during periods of simultaneous TX and RX DMA activity. The DMA_LIMIT is re-applied every time the channel acquires the bus, and does not accumulate over more than one bus acquisition.

A diagram showing the block and its association with surrounding components is below:

Figure 4-1 BIM Top Level Diagram



4.2.1 DMA Arbitration

DMA access types are:

- TX transmit buffer descriptor reads
- TX completion register writes

- TX data buffer reads
- RX free buffer descriptor ring reads
- RX completion ring writes
- RX data writes

The TX and RX units internally resolve DMA access needs such that a request for a TX completion register write does not occur while a TX data buffer read is occurring or an RX descriptor read doesn't happen during RX completion ring writes. Thus, arbitration at this block level only between the TX and RX units.

The arbiter generally works on a first come first serve basis. Should requests appear simultaneously from TX and RX, grant is given in the following priority:

1. RX descriptor fetches
2. TX descriptor fetches
3. TX or RX completion register/ring writes
4. TX or RX data buffer accesses

Should TX and RX simultaneously issue the same type access, grant is given to the unit which least recently was given bus access.

The Arbitration Register specifies in units of 64 bytes how long a burst from TX or RX can proceed before an outstanding request from the other unit is allowed to interrupt. This rearbitration feature requires the arbiter and bus logic to work in conjunction such that a bus operation that is interrupted is properly resumed. Thus, intermediate address and burst transfer progress information is maintained in the logic. Note that rearbitration occurs only during TX or RX data buffer accesses and not during descriptor accesses from either unit.

4.2.2 PIO Arbitration Priority

PIO operations have priority over DMA requests to avoid deadlocks on the I/O bus or within the bus interface logic.

Transmit Block 5

5.1 TX DMA block

The TX DMA block works with the PCI BIM block to read transmit frames from the host memory. The DMA block handles the chaining of TX frames and TCP checksum generation. In addition, the TXDMA block can be programmed to allocate network bandwidth to four Class Based Queues in host memory. The TX DMA attempts to move data on cache line boundaries between the PCI BIM block and the TX FIFO. Handling of misaligned bursts due to PCI retries is handled within the PCI BIM and is transparent to the TX DMA.

Chaining

The TX DMA block implements the gather function of transmit buffers. Proper byte alignment is performed to ensure that data bytes at multiple buffers boundary, belonging to the same frame, are packed sequentially in the TX FIFO. The chaining of transmit buffers is done on-the-fly during frame data transfer between the PCI Bus Interface and the TX FIFO.

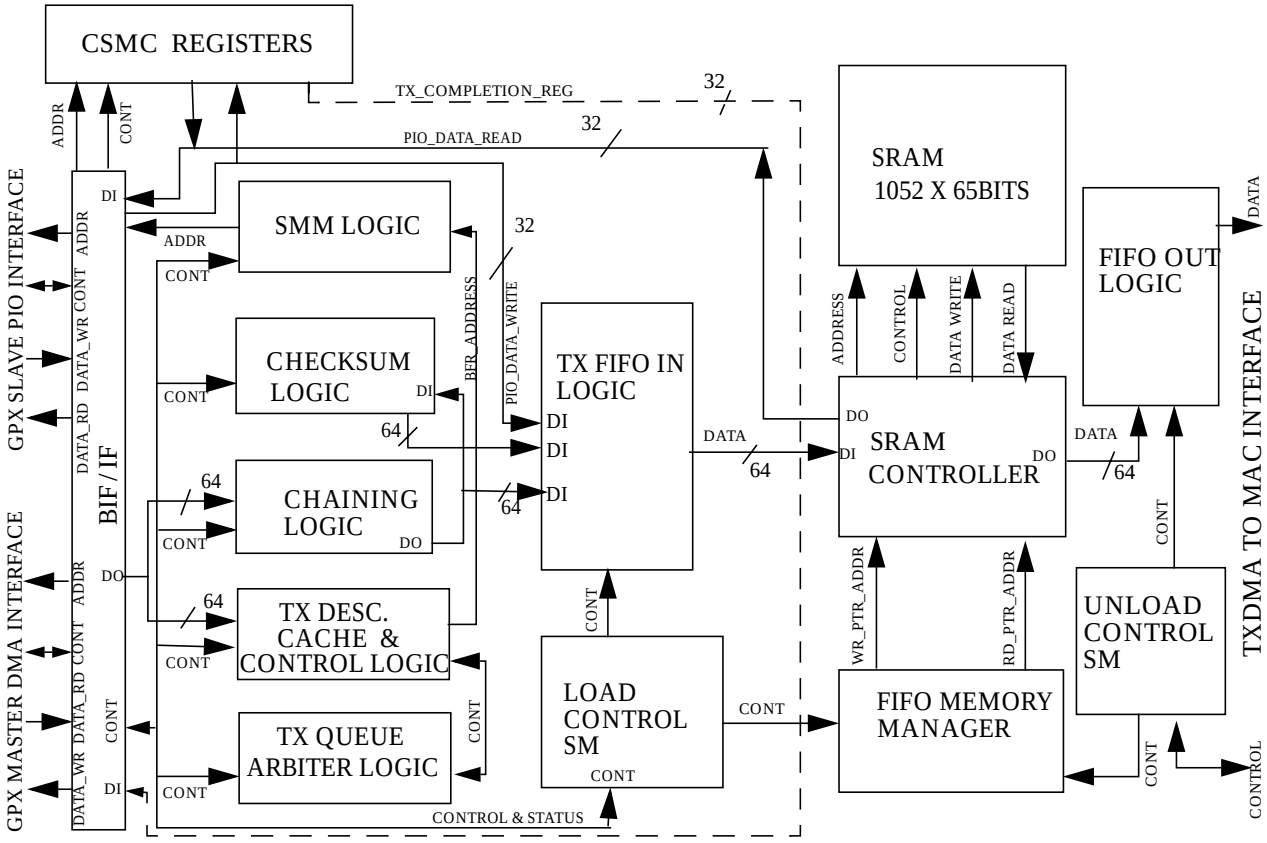
Checksum

This block provides the hardware support for TCP checksum computation in the transmit data path. This optional function can be enabled or disabled on a per frame basis through a descriptor control bit. If the function is enabled, the software has to provide, in the descriptor, a start offset and a stuff offset. It is assumed that software will initialize the TCP checksum field in the TCP frame to compensate for the fact that the hardware calculates the checksum over the actual IP header, rather than the pseudo-IP header. The 16-bit TCP checksum is computed on-the-fly during frame data transfer between the PCI Bus Interface and the TX FIFO.

Tag bits generation

As the frame is being written into the TX FIFO, end of the frame is denoted by tag bits in two consecutive words: the last data word and the control word.

Figure 5-1 TXDMA Block Diagram



The TXDMA Block is administered and monitored by the contents of the Command, Status, Mask, and Configuration (CSMC) Registers. The System Memory manager (SMM) module manages the transmit host memory data structures. The BIF / IF module provides the block level interface to the PCI Bus interface. The TX QUEUE Arbiter implements a Weighted Round Robin arbitration on the four sets of descriptor rings, each of which is associated with a bandwidth allocation and Quality of Service, progmmable through the TX QUEUE Maxburst Registers. The Load Control State Machine controls the transfer of data from the host memory to the TX FIFO SRAM, while the Unload Control State Machine controls the transfer of data from the TX FIFO to the MAC, where it is put onto the network.

5.1.1 TX QUEUE Arbiter

The TXQUEUE Arbiter consists of Wighted Round Robin Arbitration and is controllable via the contents of the MaxBurst_i Registers associated with each descriptor ring.

5.1.1.1 Weighted Round Robin Arbitration

Each queue is associated with a weight MaxBurst_i. The arbiter will serve each queue upto MaxBurst_i bytes before moving on to the next queue in that round.

The arbiter serves queue_1 first. It sums the buffer sizes, byte_count_1, of the descriptors from queue_1 as it is being processed from the descriptor cache. Packets are transmitted until the current byte_count_1 of the buffer sizes transmitted is less than or equal to Maxburst_1. If byte_count_1 is equal to Maxburst_1 when the arbiter is in the middle of a packet transmission, then the summation is allowed to go beyond the weight until the whole packet is transmitted (packet boundary), and the excessive number of bytes transmitted in that round is recorded and saved into a defecit register, Defecit_1. The arbiter then serves queue_2, queue_3, and queue_4 in a similar way. When the arbiter returns to queue_1 in the next round, the weight of the queue_1 is set to (Maxburst_1 - Defecit_1) to account for the excessive transmission of queue_1 in the previous round. This process continues again and again for all queues.

Figure 5-2 Top level flow diagram of WRR

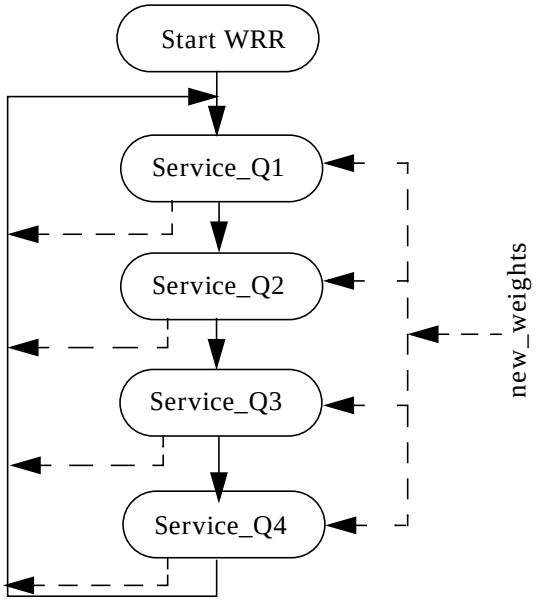
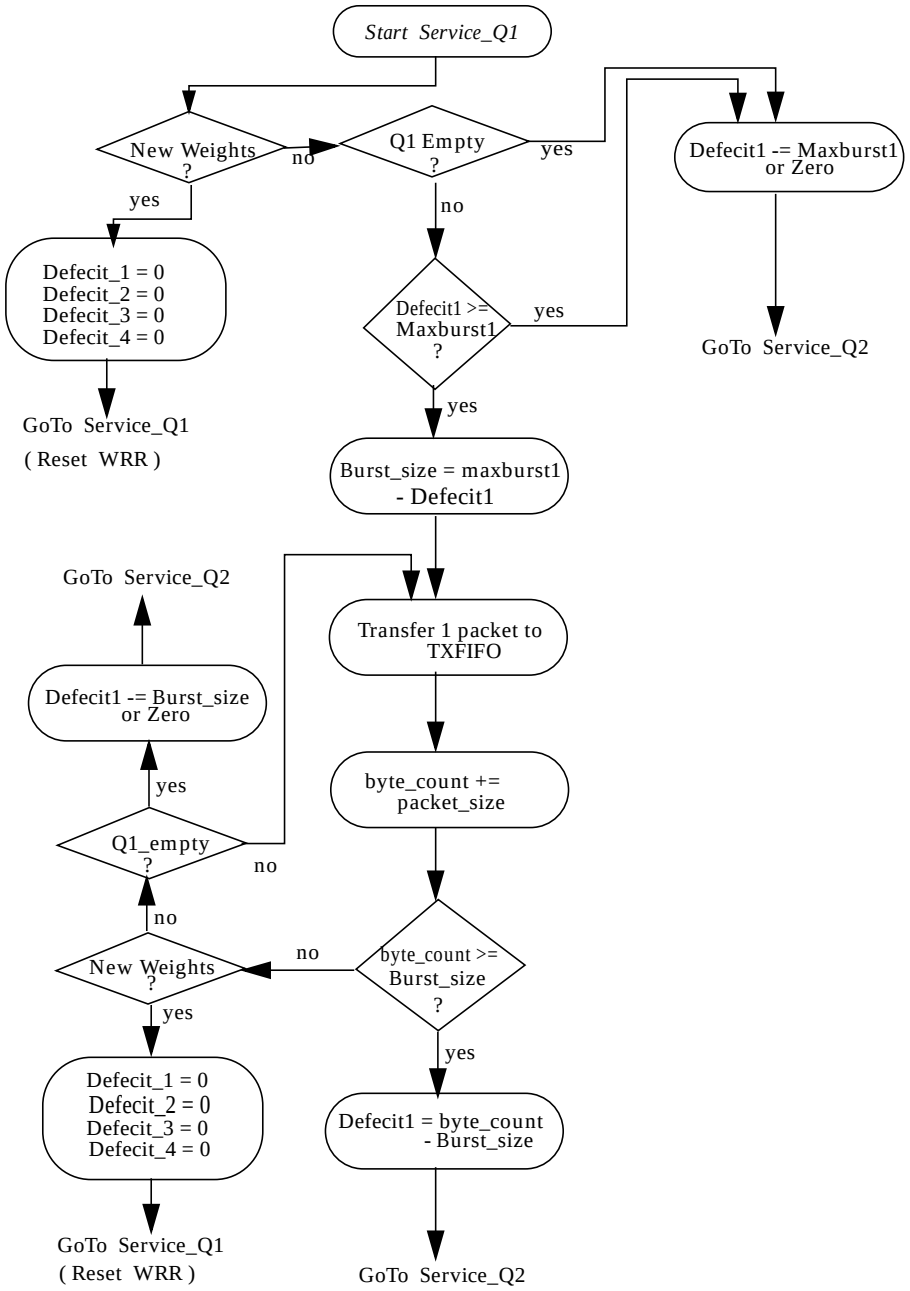


Figure 5-3 Flow diagram of Service_Q1[2,3,4]



5.2 TX Jumbo Frame and TX DMA Fifo

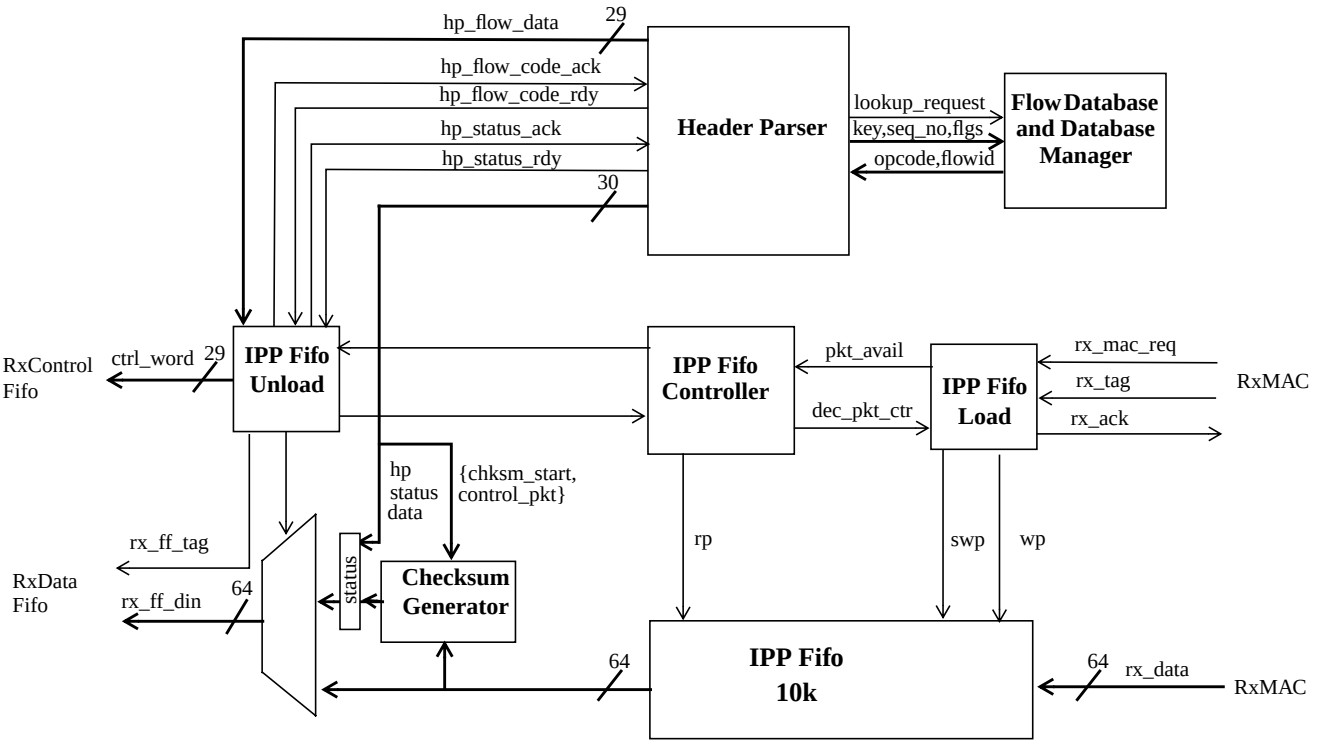
TX Jumbo Frame support and the TX DMA Fifo are dicussed in the RX DMA Block chapter.

6.1 RX DMA block

The RX DMA block works with the PCI BIM block to move receive frames into the host memory. The RX DMA block handles TCP checksum generation. The RX DMA attempts to move data on cache line boundaries between the RX FIFO and the PCI BIM block. Handling of misaligned bursts due to PCI retries is handled within the PCI BIM and is transparent to the RX DMA.

6.1.1 Rx Load

Figure 6-1 Rx Load Block Diagram



The Rx Load consists of the core datapath, called the Input Port Process (IPP) and the two submodules which perform the Layer 3 and 4 specific functions. These are called the Header Parser and Flow Database Manager (FDBM). The Rx Load interfaces with the Rx MAC and transports data from there to the Rx FIFO. The packet is rewound if it is labelled “aborted” by the MAC. In addition to basic packet transfer, the Rx Load performs the front end analysis of the packet to aid the reassembly in the Rx Load which gets passed to the HRP in the form of a control packet. After loading the packet data into the Rx FIFO the IPP core generates the control packet and adds it to the Rx Control FIFO.

In order to maintain 1 Gb/s throughput, processing of the packet from the MAC to the Rx FIFO needs to occur in 84 clocks since the worst case scenario is when MAC receives the 64 byte packets back to back with minimum IPG between them (64 bytes of data + 12 byte of IPG + 8 bytes of preamble/SFD). The Rx load should be designed to handle a packet as small as 8 bytes however. The MAC will filter anything smaller.

Using the 84 bytes as a boundary we can create a bandwidth budget for the IPP. See Table 6-1.

Table 6-1 Bandwidth Budget in the Rx Load

Sub Rx Load Processing	**Budget (clocks)**
Transfer data from the IPP FIFO to the Header Parser	10
Handshake for HP status and flow code	4
Parse and post process the results Perform the flow lookup and opcode generation	51
Transfer the packet from the IPP FIFO to the Rx FIFO	8
Perform data wrapping for checksum generation and status word generation	4
Enter the Control Packet into the Control Fifo	0
margin	7
total	84

Random Early Detection

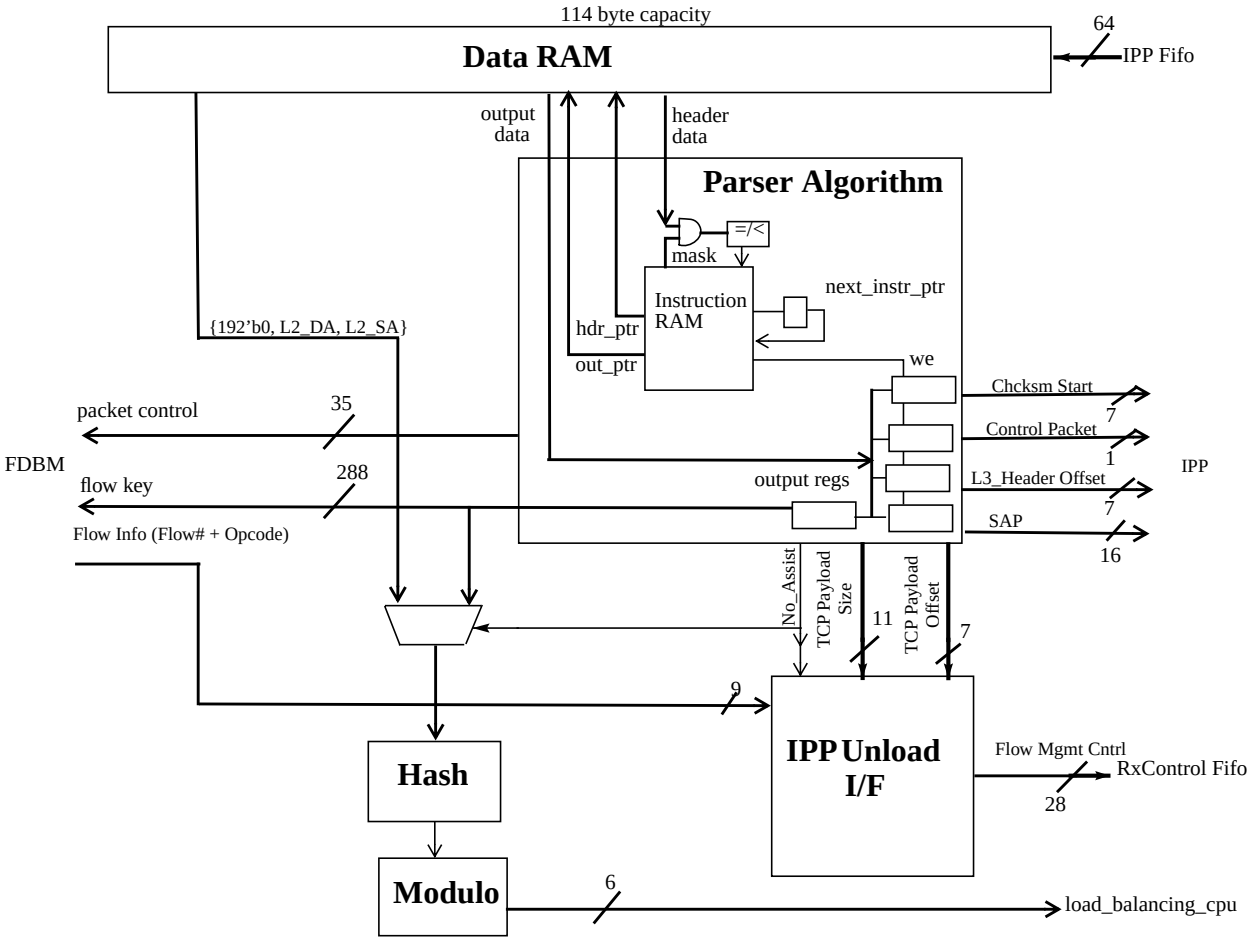
The Random Early Detection method is used to provide congestion avoidance. When the queue size exceeds a preset threshold, the Rx Load marks each packet with a certain probability for drop, where the exact probability is a function of the queue size. There are four such thresholds, each with a drop probability and the probability increases as the fifo level increases. The RED method has no bias against bursty traffic as does tail drop. It also avoids the problem of many connections simultaneously reducing their window sizes by spreading out the packet losses. During congestion, the probability of traffic

sources getting their packets dropped is proportional to the amount of traffic they produce. Each of the four thresholds has an enable bit to turn off packet marking.

Control packets identified by the Header Parser are not counted in the statistics and are not candidates for drop.

6.1.1.1 Header Parser

Figure 6-2 Header Parser Block Diagram



As shown in Figure 6-2, the Header Parser will contain two rams, one for up to 114 bytes of data, and the other containing the parsing instruction set. The parser moves through the packet header and data stored in the data ram using the instruction set preloaded into the instruction ram. The instruction set governs a series of checks which will indicate whether the packet is suitable for Layer 3 and 4 hardware assistance. The instruction set also controls the storage of key information from the packet header to be used by other submodules in Rx Load.

Data per packet always passes through the Header Parser even if there is insufficent information to parse. Appropriate information per packet will always be passed back to the IPP regardless of packet size. This information

includes checksum start offset to be used by the IPP checksum calculations. The checksum will be caculated to the end of the packet. The checksum will be stuffed into the status word attached to the end of the packet in the main Rx fifo. The Layer 3 Header offset and the SAP information extracted during parsing also form part of the status word.

If a flow key can be extracted from the packet, the Header Parser will request flow lookup from the FDBM. What it receives in return from the FDBM is the opcode and flow id. These together with the TCP Payload Offset, TCP Payload Size, no assistance flag, and small size flag are loaded into the control fifo. This control information can not be put into the control fifo until all the packet data has been transferred into the main Rx data fifo. This is because the presence of a packet in the control fifo is used as a trigger to unload data from the main Rx FIFO.

The programmability of the instruction RAM adds flexibility in the parsing and expandability. If necessary it can be programmed to parse down another OSI level or be updated to reflect changes to networking or transport protocols. An instruction is read each cycle. If required, specific header data is saved during the instruction. The next instruction pointer is taken from the current instruction.

The following table details instructions which will be loaded into the Instruction RAM. These will be used to Parse packets to Layer 4.

Table 6-2 Header Parser Instruction Table

Instruction Number	**Instruction Name**	**Description**	**Instruction = {Mask,Compare Value,Operator,Success next offset, Success next instruction, Fail next offset, Fail next instruction, Output opcode, Output argument, Output enable, outshift, outmask}**
0	Packet Arrival (S1_PCKT)	Check for new packet arrival.	0xffff, 0x0000, OP_NP, 6, S1_VLAN, 0, S1_PCKT, CL_REG, 0x3ff, 1, 0x0, 0x0000
1	VLAN (S1_VLAN)	Check type field for VLAN type. If it is a VLAN type, the parser must check the CFI bit.	0xffff, 0x8100, OP_EQ, 1, S1_CFI, 0, S1_8023, IM_CTL, 0x00a, 3, 0x0, 0xffff
2	CFI (S1_CFI)	Check if the CFI bit is set. If set, current microcodes try to ignore that and pass to host to handle that. If dicide to drop then the “success next instruction should set to S1_DROP.	1) Ignore CFI pkt and let host handle it (default) 0x1000, 0x1000, OP_EQ, 0, S1_CLNP, 1, S1_8023, CL_REG, 0x000, 0, 0x0, 0x0000 2) Drop CFI pkt if found. 0x1000, 0x1000, OP_EQ, 0, S1_DROP, 1, S1_8023, CL_REG, 0x000, 0, 0x0, 0x0000
3	802.3 (S1_8023)	The type/length field is examined to determine if it is a 802.3 packet or a “traditional” Ethernet packet. If it is an 802.3 packet, the type field must be checked to ensure it is an 802.3 “Ethernet” packet.	0xffff, 0x0600, OP_LT, 1, S1_LLC, 0, S1_IPV4, CL_REG, 0x000, 0, 0x0, 0x0000

Table 6-2 Header Parser Instruction Table

Instruction Number	Instruction Name	Description	Instruction = {Mask,Compare Value,Operator,Success next offset, Success next instruction, Fail next offset, Fail next instruction, Output opcode, Output argument, Output enable, outshift, outmask}
4	LLC (S1_LLC)	The first part of the snap field is checked to see if it matches that of a 802.3 “Ethernet” packet.	0xffff, 0xaaaa, OP_EQ, 1, S1_LLCc, 0, S1_CLNP, CL_REG, 0x000, 0, 0x0, 0x0000
5	LLC CONT. (S1_LLCc)	The second part of the snap field is checked to see if it matches that of a 802.3 “Ethernet” packet. If it is not, there is no hardware assist.	0xff00, 0x0300, OP_EQ, 2, S1_IPV4, 0, S1_CLNP, CL_REG, 0x000, 0, 0x0, 0x0000
6	IPV4 (S1_IPV4)	The Layer 3 header type is checked to see if it is an IPv4 header. If not, it will be checked to see if it is an IPv6 header.	0xffff, 0x0800, OP_EQ, 1, S1_IPV4c, 0, S1_IPV6, LD_SAP, 0x100, 3, 0x0, 0xffff
7	IPV4 CONT. (S1_IPV4c)	Although the IPv4 type matched, another check is made to determine if the version number in the header is also 4. If not, there is no hardware assist.	0xff00, 0x4500, OP_EQ, 3, S1_IPV4F, 0, S1_CLNP, LD_SUM, 0x00a, 1, 0x0, 0x0000
8	IPV4 FRAGMEN TATION (S1_IPV4F)	Check that the IP segment is not a fragment. If a fragment, there is no hardware assist. Save the flow id into a register.	0x3fff, 0x0000, OP_EQ, 1, S1_TCP44, 0, S1_CLNP, LD_LEN, 0x03e, 1, 0x0, 0xffff
9	TCP WITH IPV4 (S1_TCP44)	Check that the transport protocol is TCP. If not, there is no hardware assist. Save the remainder of the flow id into a register.	0x00ff, 0x0006, OP_EQ, 7, S1_TCPSQ, 0, S1_ESP4, LD_FID, 0x182, 1, 0x0, 0xffff
10	IPV6 (S1_IPV6)	The Layer 3 header type is checked to see if it is IPv6. If not, there is no hardware assist.	0xffff, 0x86dd, OP_EQ, 1, S1_IPV6L, 0, S1_CLNP, LD_SUM, 0x015, 1, 0x0, 0x0000
11	IPV6 len (S1_IPV6L)	Although the IPv6 type matched, another check is made to determine if the version number in the header is also 6. If not, there is no hardware assist. Save the flow id into a register.	0xf000, 0x6000, OP_EQ, 0, S1_IPV6c, 0, S1_CLNP, IM_R1, 0x128, 1, 0x0, 0xffff
12	IPV6 CONT. (S1_IPV6c)	IPV6 check continue	0x0000, 0x0000, OP_EQ, 3, S1_TCP64, 0, S1_CLNP, LD_FID, 0x484, 1, 0x0, 0xffff
13	TCP WITH IPV6 (S1_TCP64)	Check that the transport protocol is TCP. If not, there is no hardware assist. Save the remainder of the flow id into a register.	0xff00, 0x0600, OP_EQ, 12, S1_TCPSQ, 0, S1_ESP6, LD_LEN, 0x03f, 1, 0x0, 0xffff

Table 6-2 Header Parser Instruction Table

Instruction Number	Instruction Name	Description	Instruction = {Mask,Compare Value,Operator,Success next offset, Success next instruction, Fail next offset, Fail next instruction, Output opcode, Output argument, Output enable, outshift, outmask}
14	TCP SEQUENCE NO. (S1_TCPSQ)	Save the TCP sequence number in a register.	0xFFFF, 0x0080, OP_EQ, 0, S2_HTTP, 0, S1_TCPFG, LD_SEQ, 0x081, 3, 0x0, 0xffff
15	TCP control flags (S1_TCPFG)	Load TCP flags	0xFFFF, 0x8080, OP_EQ, 0, S2_HTTP, 0, S1_TCPHL, ST_FLG, 0x145, 2, 0x0, 0x002f
16	TCP length (S1_TCPHL)		0x0000, 0x0000, OP_EQ, 0, S1_TCPHc, 0, S1_TCPHc, LD_R1, 0x205, 3, 0xB, 0xf000
17	TCP length cont (S1_TCPHc)		0x0000, 0x0000, OP_EQ, 0, S1_PCKT, 0, S1_PCKT, LD_HDR, 0x0ff, 3, 0x0, 0xffff
18	Cleanup (S1_CLNP)	reset regsiters step1	0x0000, 0x0000, OP_EQ, 0, S1_CLNP2, 0, S1_CLNP2, IM_CTL, 0x001, 3, 0x0, 0x0001
19	Cleanup 2 (S1_CLNP2)	reset regsiters step2	0x0000, 0x0000, OP_EQ, 0, S1_PCKT, 0, S1_PCKT, CL_REG, 0x002, 3, 0x0, 0x0000
20	Drop packet (S1_DROP)		0x0000, 0x0000, OP_EQ, 0, S1_PCKT, 0, S1_PCKT, IM_CTL, 0x080, 3, 0x0, 0xffff
21	No HTTP (S2_HTTP)		0x0000, 0x0000, OP_EQ, 0, S1_PCKT, 0, S1_PCKT, IM_CTL, 0x044, 3, 0x0, 0xffff
22	IPV4 ESP encrypted (S1_ESP4)	check if this is a IPV4 ESP pkt	0x00ff, 0x0032, OP_EQ, 0, S1_CLNP2, 0, S1_AH4, IM_CTL, 0x021, 1, 0x0, 0xffff
23	IPV4 AH encrypted (S1_AH4)	check if this is a IPV4 AH pkt	0x00ff, 0x0033, OP_EQ, 0, S1_CLNP2, 0, S1_CLNP, IM_CTL, 0x021, 1, 0x0, 0xffff
24	IPV6 ESP encrypted (S1_ESP6)	check if this is a IPV6 ESP pkt	0xff00, 0x0032, OP_EQ, 0, S1_CLNP2, 0, S1_AH6, IM_CTL, 0x021, 1, 0x0, 0xffff
25	IPV6 AH encrypted (S1_AH6)	check if this is a IPV6 AH pkt	0xff00, 0x0033, OP_EQ, 0, S1_CLNP2, 0, S1_CLNP, IM_CTL, 0x021, 1, 0x0, 0xffff

Register descriptions:

DADDR[5:0] : current data address. DADDR resets to 6 not zero.

D1[15:0] and D2[15:0] are the output data from the data RAM where parsing data been stored.

flowoff[4:0] : current offset within flowid.

flowid[287:0] : flowid. Loaded from the packet data or immediate.

seqoff[1:0] : current offset within seqno.

seqno[31:0] : TCP sequence number.

control[3:0] : bit 3 is the no_assist bit, bits[2:0] are the TCP control bits. Loaded from packet data or immediate.

SAP[15:0] : ethertype from the packet. Loaded from packet data or immediate.

R1[6:0] : An accumulator. Updated from packet data or immediate. Used to generate new DADDR or DADDR-based registers.

flags[1:0] : TCP control flags.

control[5:0] : FDBM control flags.

payloadlen[15:0]: Payload length in bytes., used to compute FDBM control flags.

L3_offset[6:0] : Offset to the L3 header, in 16 bit words. Loaded based on DADDR.

csum_start[6:0] : offset to start of checksum in 16 bit words. Loaded based on DADDR.

hdr_split[6:0] : location to split header buffer, in 16 bit words.

Instruction fields definitions:

mask[15:0] : Mask to apply to data from packet.

val[15:0] : value to compare data from packet against.

op[1:0] : Opcode for data and val, defined below.

soff[6:0] : The success next offset register. If match, soff[5:0] is the offset to the new location in the packet from the current DADDR. If soff[6] os 1 then R1 is not added into the new offset.

snext[4:0] : The success next instruction. If match, next instruction to execute.

foff[6:0] : The fail next offset register. If not match, foff[5:0] is the offset to the new location in the packet from the current DADDR. If foff[6] is 1 then R1 should also be added into the new offset. If foff[6] is 0 then R1 is not added into the new offset.

fnext[4:0] The fail next instruction. If not match, next insturction to execute.

outop[3:0] : Output opcode to execute, defined below.

outarg[11:0] : Argument to outop, defined with each opcode.

outenab[1:0]: If outenab[0] is 1 , outop will always be executed, if outenab[1] is set, outop will be executed if not match. If both bits are set outop will always be executed, if neither bit is set outop will not be executed.

outshift[3:0] : number of bits to right-shift D2 before input to a register. This is a barrel shift, bits which shift off the right side of the register reappear on the left.

outmask[15:0] : Mask to apply to D2 before input to register.

Opcode definitions:

OP_EQ: (D1 & mask) == val

OP_LT: (D1 & mask) < val

OP_GT : (D1 & mask) > val

OP_NP: True when a new packet is waiting to be parsed, false otherwise.

Outopcode definitions:

1. CLR_REG: Selectively reset registers. If outarg[0] is set then reset DADDR. If outarg[1] is set then reset flowid and flowoff. If outarg[2] is set then reset seqoff and seqno. If outarg[3] is set then reset sap. If outarg[4] is set then reset R1. If outarg[5] is set then reset control and flags. If outarg[6] is set then reset L3off. If outarg[7] is set then reset csumstart. If outarg[8] is set then reset hdrsplit. If outarg[9] is set then reset payloadlen.

2. LD_FID : Load the flowid register at the current flowoff. outarg[5:0] is the offset from DADDR to load. outarg[10:6] is the length in 16 bit words to load (maximum 18, because the flowid is 288 bits wide).

3. LD_SEQ: Load the seqno at the current seqoff. outarg[5:0] is the offset from DADDR to load, outarg[7:6] is the length in 16 bit words to load.

4. LD_CT: Load the control field. outarg[5:0] is the offset from DADDR to load.

5. LD_SAP: Load the SAP. outarg[5:0] is the offset from DADDR to load. If outarg[8] is 1, also load the L3offset field with (DADDR + outarg[6:0] +1).

6. ACC_R1: Accumulate R1. outarg[5:0] is the offset from DADDR to load.

7. LD_L3: If outarg[9] is 1, load the L3_offset field from outarg[6:0]. If outarg[9] is 0, load the L3_offset from DADDR. outarg[6:0] is the amount to add to DADDR and if outarg[7] is set, also add R1.

8. LD_SUM: If outarg[9] is 1, load the csum_start field from outarg[6:0]. If outarg[9] is 0. Load the csum_start from DADDR. outarg[6:0] is the amount to add to DADDR and if outarg[7] is set, also add R1.

9. LD_HDR: If outarg[9] is 1, load the hdr_split from outarg[6:0]. If outarg[9] is 0, load the hdr_split field from DADDR. If outarg[8:7] is 00, add 0 to the result. If outarg[8:7] is 01, also add R1. outarg[8:7] is 10, also add 7'h1. If outarg[8:7] is 11, the result is undefined.

10. IM_FID: Load the flowid register at the current flowoff with a single 16 bit word with the immediate data from {5'b0, outarg[10:0]}.

11. IM_SEQ: Load the seqno register at the current seqoff with a single 16 bit word with the immediate data from {5'b0, outarg[10:0]}.

12. IM_SAP: Load the SAP with {5'b0, outarg[10:0]}.

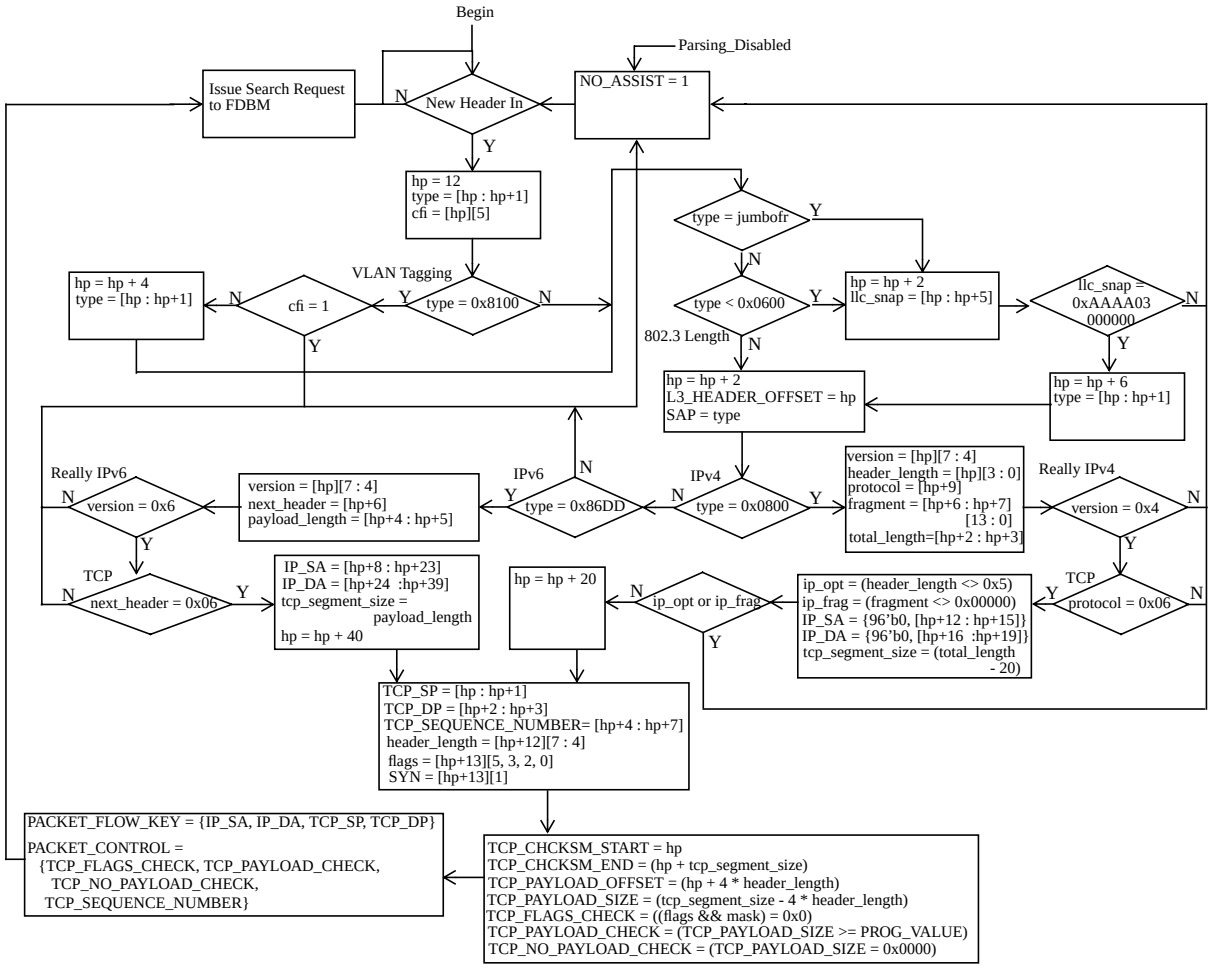
13. IM_R1: Accumlate R1 with outarg[6:0].

14. IM_CTL: Load the control register from outarg[4:0].

15. LD_LEN: Load payloadlen. outarg[5:0] is the offset from DADDR to load.

16. ST_FLG: Set the flags register.

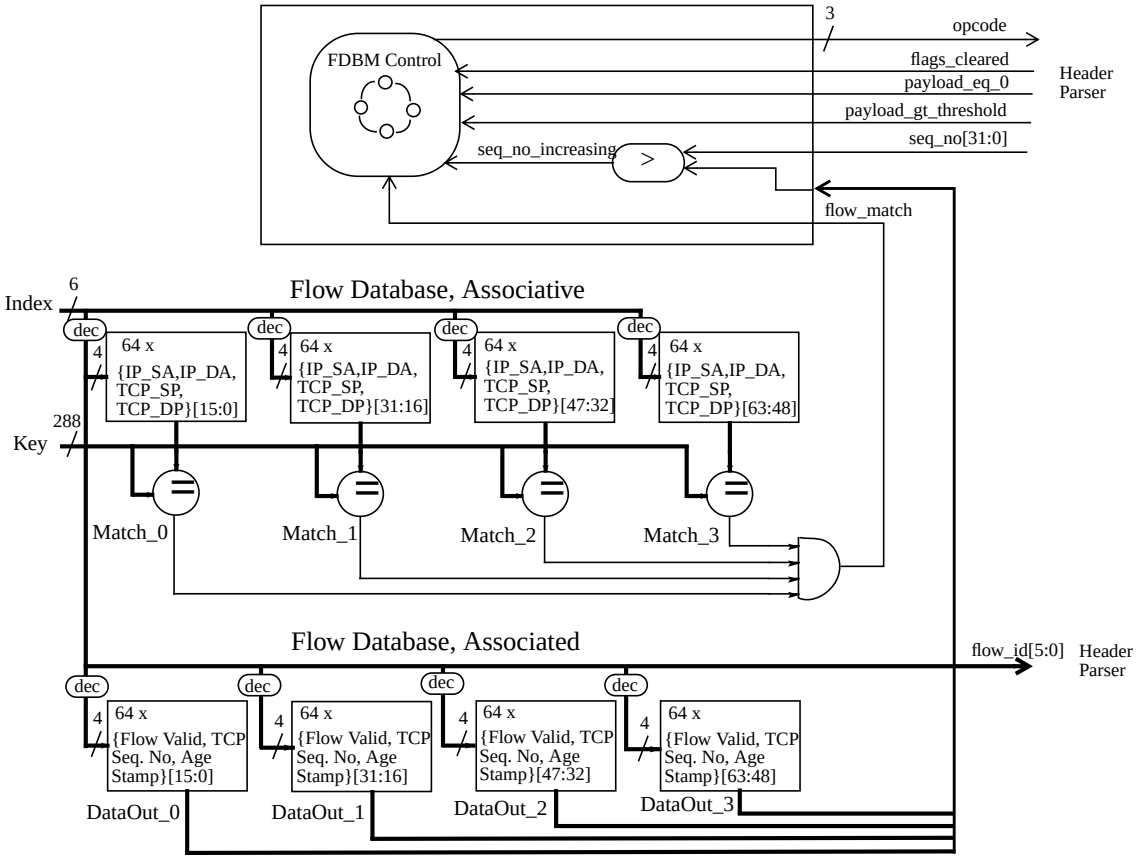
Figure 6-3 Header Parser Flow Chart



Note: If the mask is set to 0x2F then only the ACK bit will be masked. This would be one “normal” setting of the mask.

6.1.1.2 Flow Database Manager

Figure 6-4 Flow Database and Flow Database Manager Block Diagram



The Flow Database Manager (FDBM) is responsible for generating an opcode for use by the Host Receive Process (HRP). These opcodes will indicate the type of packet that needs to be processed and what type of hardware assistance is given to it. The FDMB interfaces with the Header Parser, passing information back to it to be incorporated into the control packet. The basic flow of data is as follows.

1. The Header Parser makes a request for flow lookup and opcode generation by passing to the FDBM the 288 bit flow key based on the IP source and destination addresses and the TCP source and destination ports.

2. Based on this key, the FDBM then searches through a CAM or a four-way RAM structure to provide flow match information within 16 cycles.

3. If the key does not exist and the packet satisfies flow setup criteria, it will be entered into the table. If necessary, the oldest flow will be removed from the table to make room for this new flow

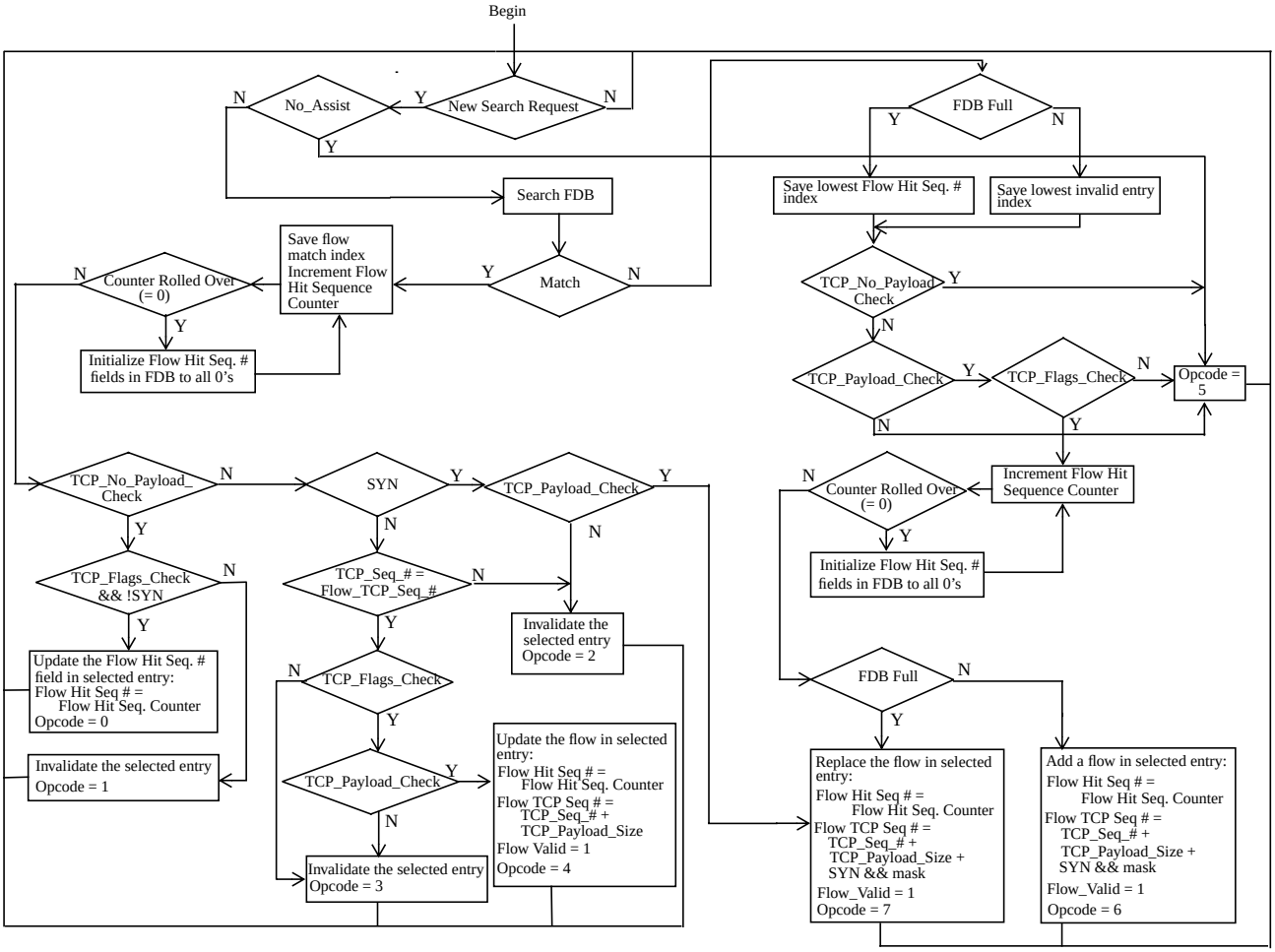
4. A flow age stamp is captured into the database upon initialization of the table for that flow. This enables the FDBM to search among the entries for the oldest flow.

5. To determine the opcode associated with the packet, checks are done on the TCP payload size, the TCP sequence number and the TCP flags. Table 6-3 shows a summary of criteria to generate each opcode and its affect at the HRP. Section Figure 6-5, “Flow Database Manager Flow Chart shows the details of decision making for the FDBM.

6. The opcode and flow id are passed back to the Header Parser.

While the FDBM is searching for a flow match, it will also be comparing flow hit sequence numbers to find the smallest (i.e. the oldest) valid sequence number or the smallest invalid entry. This index will be used in the event that a new flow needs to be entered into the table. The age stamp is incremented once per flow match and once per flow initialization. Occasionally the flow age counter will reach its terminal count and roll over. If this occurs the flow age stamps are reinitialized to zero. This creates some unfairness in flow replacement since the true relative ages are lost. However, this simplifies implementation at the cost that unfair replacemnt for up to 64 flows occurs every 48 minutes - 19 hours.

Figure 6-5 Flow Database Manager Flow Chart



Control Opcodes

The Flow Database Manager (FDBM) within the IPP evaluates an incoming packet header as characterized prior by the header parser. The FDBM first identifies and labels with an opcode packets which are not TCP or could not be successfully parsed beyond Layer 3. TCP packets are searched in the flow database to see if they are part of an already established TCP flow. They are characterized and processed as described in the FDBM algorithm which categorizes TCP packets in one of eight ways. The list below outlines the categorization and what processing the hardware must do in each case:

Table 6-3 Opcode Definition

Opcode	Criteria for Selection	Instruction to HRP
0	TCP control packet with no flags set. Flow was previously established. An example situation would be a TCP ACK.	Do not set up the HRP flow. Do not tear down the flow. Do not reassemble since there is no data. Put the whole packet, all of which is header, in a 256B buffer. This packet is associated with its flow.
1	TCP control packet with one or more control flags set. Flow was previously established. A SYN bit was detected, which is an error, or one of the TCP flags is set which makes this packet a case better handled in the slow track.	Tear down the HRP flow. Do not set up a new flow. Do not reassemble since there is no data. Put the whole packet, all of which is header, in a 256B buffer. This packet is associated with its flow.
2	TCP packet whose sequence number does not follow that of the previous TCP packet in the stream or the packet has less than the programmable threshold of TCP payload with the SYN bit set. This is interpreted as a new connection but which doesn't require reassembly. Jumbo packets are assigned this opcode by the IPP (overriding the opcode given by the FDBM).	Tear down the HRP flow. Do not set up a new flow. Do not reassemble this packet with the flow. Put the whole packet into a 2k size buffer.
3	Last TCP packet in the flow. The packet passes the flags and sequence number check except it has less than the programmable threshold size worth of TCP data.	Reassemble data with the flow. Tear down the HRP flow. Do not set up a new flow.
4	The packet passes all tests for a normal flow reassembly packet.	Reassemble data with the flow. Do not set up a flow because it should already exist. Do not tear down the flow.

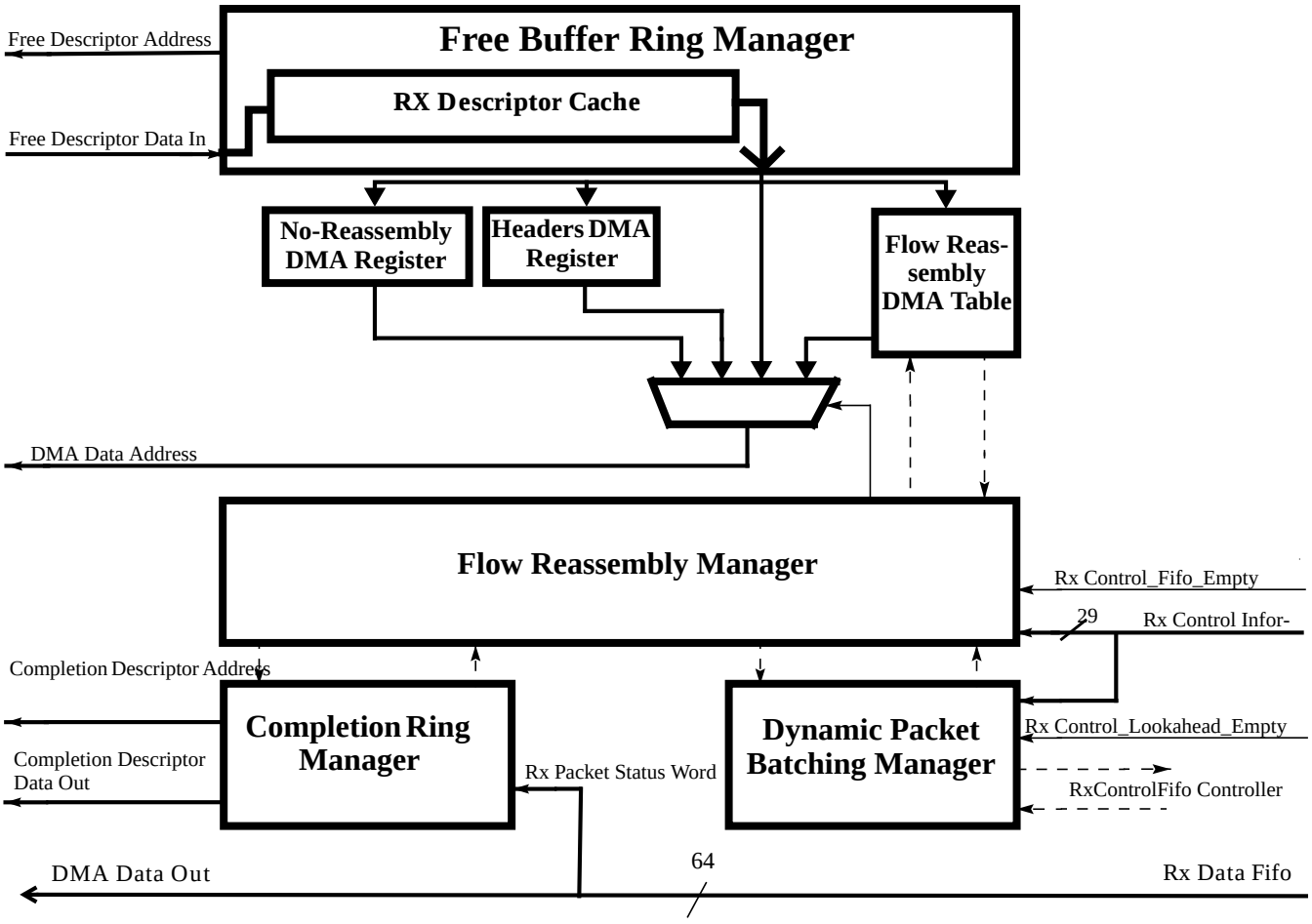
Table 6-3 Opcode Definition

Opcode	Criteria for Selection	Instruction to HRP
5	Non-reassembleable TCP packet (possibly too short or TCP flags set) or non TCP packet. This is a stand-alone packet so there is no flow established or affiliated with it. The packet has either no data or some flags have been set, making it a case better handled in the slow track. The SYN bit is not checked since the flow setup occurs upon receipt of packets with data.	No HRP flow setup. No flow tear down (there shouldn't be a flow).
6	First TCP packet of a new flow being established. Packet has passed all checks for a normal flow reassembly packet so a new flow shall be created.	Set up a new HRP flow using the flow id. Reassemble data into the new flow.
7	First TCP packet of a new flow but flow database already full with 64 flows previously established. Same as opcode 6 except due to lack of open entries in the flow table, an old flow must be replaced. This opcode can also be generated by a packet with greater than a threshold of data arriving on an existing flow with a SYN bit set. This is interpreted as a new connection which requires reassembly.	Tear down old flow. Replace it with a new HRP flow using the flow id. Reassemble data into the new flow.

6.1.2 HRP

This section describes the final major functional block in the receive path prior to the bus interface logic. The Host Receive Process (HRP) block's basic purpose is to remove from the RX Fifo packets from TCP streams to be reassembled as well as other network traffic packets and efficiently get them into buffers in host memory.

Figure 6-6 HRP Block Diagram



6.1.2.1 Basic Flow

Refer back to Figure 2-1 as needed for the overall block diagram of the chip. Figure 6-6 shows the basic HRP block diagram.

As detailed in the sections covering the Input Port Process (IPP) and Flow Data Base Manager (FDBM), after a lot of processing by those modules, RX packets sent to Cassini are placed into the RX Data Fifo with a corresponding entry providing information on the packet placed into the RX Control Fifo. The RX Fifos absorb much of the varying response time of the host system.

On the system side, the driver has set aside empty memory buffers waiting to receive incoming packets. These buffers are referenced through a circular queue - the Free Buffer Descriptor Ring - maintained in host memory. Logic

within the HRP has the duty of fetching a descriptor entry which points to a free buffer with a goal of having the descriptor on hand prior to it being needed by an incoming packet thus preventing a stall while the descriptor fetch is made. The descriptor ring contains memory buffers for three types of packets. Each buffer is one page in size. Small packets 256 - swivel_offset bytes and smaller not part of a reassembly stream get packed into a buffer. Packets larger than 256 - swivel_offset bytes not being reassembled into a TCP stream are packed into a buffer allocated for large non-reassembled packets. TCP packets being reassembled into a stream go into separately allocated reassembly buffers.

As each packet gets placed into host memory through HRP processing, an update of the RX Completion Ring is made to provide the driver sufficient information to process the buffer which the packet was written into.

The HRP monitors the RX Control Fifo to see if a new packet has been deposited. Once a packet has been detected, the HRP control logic proceeds to extract information from the control fifo and data fifo status word which will determine which of a number of possible ways the HRP processes the packet on its way to host memory.

The basic HRP block diagram is shown in Figure 6-6. Control logic governing RX descriptor ring and RX completion ring access as well as RX Control and Data Fifo access is in the Flow Reassembly DMA Controller block. That block also regulates movement of packet data into the bus interface fifo for final deposit into host memory.

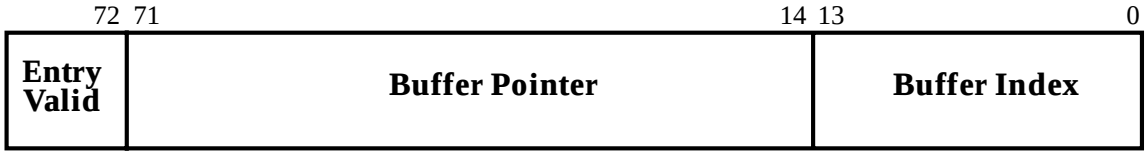
6.1.2.2 RX Descriptor Caching

To keep the process of packet transfer from the RX Fifo into host memory from slowing down due to lack of descriptors pointing to free buffers, the H/W should proactively get descriptors from the descriptor ring and save them on chip until they are needed. RX free buffer ring descriptors are spaced 16 bytes apart on the descriptor ring so in 64 byte cacheline systems efficient access of at least four at once can be made. The general characteristics of the RX descriptor cache are as follows:

- The cache is essentially two separate caches but with one set of control logic to manage descriptor fetches, format descriptor data into a 72 bit word, monitor the number of cache entries empty, invalidate an entry after H/W completes use of it, and track descriptors available from the driver
- For Cassini+, there are 2 rx_free_descriptor caches, one for non_encrypted packets, the other for encrypted ones. Each rx_free_descriptor cache has 36 bytes to store up to 4 free buffer descriptors.
- Whenever four or more entries in the cache are empty, the cache control logic fetches four descriptors if the RX Kick Register indicates descriptors are available and places them into the appropriate portion of the cache.

The format of each entry in the RX Descriptor Cache is as follows:

Figure 6-7 Rx Descriptor Cache Entry



As in GEM, if the driver does not post new descriptors faster than Cassini needs them, an interrupt (Rx_Buf_NotAv) is generated. The H/W detects this condition by testing to see if the kick register value equals the number of the descriptor last read from the descriptor ring.

6.1.2.3 RX Fifos Control Fields

The HRP control logic makes its decisions about processing RX Data Fifo contents based on control information carried in the RX Control Fifo status and RX Data Fifo Packet Status word. The fields of these two entities are described next.

6.1.2.4 RX Control Fifo Information

Pointer logic affiliated with the read and write ports of the RX Control Fifo couple the Input Port Process which places information into the RX Fifo and Host Receive Process which extracts it. Each packet regardless of size consumes one 29 bit wide entry within the RX Control Fifo. Whenever the control fifo is not empty, the HRP logic has at least one packet requiring processing. Each entry within the RX Control Fifo has the following fields:

SAP	L3 Hdr offset	Encrypt-ed packet	Load Balancing CPU No.	no assist	force flow lookahead	opcode	cksm start	TCP Pload Offset	TCP Pload Size
[64:49]	[48:43]	[42]	[41:36]	[35]	[34]	[33:31]	[30:24]	[23:17]	[16:3]

jumbo header split enable	jumbo pkt	small pkt
[2]	[1]	[0]

- **SAP** - Secondary Access Point which indicates the ethertype to the driver so it can send the packet to the correct stream.
- **Layer3 Header Offset** - Offset in bytes of the first byte of the IP header.
- **Encrypted Packet** - If set, the Tcp packet is encrypted.

- **RLoad Balancing** - Hash derived value the driver can use to distribute the RX packet processing load.
- **No Assist** - If set indicates the IPP parser was unable to successfully parse this packet. The packet is not part of a TCP stream H/W will be reassembling.
- **Opcode** - The Flow Database Manager within the IPP categorizes the packet into one of eight criteria. This opcode is used by the HRP to process the packet on its way into host memory. The opcodes are described later.
- **Checksum Start** - This is the offset value into the packet from which Rx checksumming began. It is in 2 byte granularity.
- **TCP Payload Offset** - A pointer to the beginning byte of the TCP payload in the packet. The HRP uses this to do header splitting.
- **TCP Payload Size** - Indicates the length of the TCP Payload. The HRP uses this to do reassembly.
- **Jumbo Header Split Enable** - This indicates that header splitting will apply to the packet if it is a jumbo sized packet.
- **Jumbo Packet** - If the entire packet is 1522 bytes or greater this bit is set. Among other things, this informs the HRP to pack the packet into one or more page size free buffers.
- **Small packet** - If the entire packet is 256 bytes or less this bit is set. Among other things, this informs the HRP to pack the packet into one 256 byte cell of a page size free buffer.

6.1.2.5 RX Data Fifo Packet Status Word Information

Each packet within the RX Data Fifo has a 64 bit Packet Status word with the following fields:

- **Rx_data_fifo[63] : Abort** - Bit set if the MAC determined this packet is to be aborted by the IPP.
- **Rx_data_fifo[62] : Bad** - Bit set if the MAC determined this to be a bad packet (CRC error, etc.).
- **Rx_data_fifo[61] :** reserved.
- **Rx_data_fifo[60] : Address Filter Pass - Indicates that one of the bits in the Perfect Address Match Pass field is set.**
- **Rx_data_fifo[59:44] : Address Hash Value** - The multicast address hash value to be used by the driver to do a second level of address filtering.
- **Rx_data_fifo[33:30] : Perfect Address Match Pass** - The packet matched one of the 16 multicast registers as indicated in this status word.
- **Rx_data_fifo[29:16] : Packet Length** - The length of the entire packet in bytes.
- **Rx_data_fifo[15:0] : TCP Checksum Value** - Calculated checksum over the packet if it is non-reassembly . Calculated checksum over TCP payload length if it is a reassembly packet or jumbo packet with header split.

Checksum Considerations

The checksum is assumed by Cassini to be a 16-bit field used by some upper layers, like TCP, to verify the integrity of frames received. Given that checksum is not defined at the MAC level, the details of checksum generation/checking vary as a function of the specific upper layer sourcing or consuming the frame. Cassini's checksum algorithm is specific to the TCP transport layer protocol.

For receive frames, a checksum is computed on even byte boundaries, and made available to the host as part of the frame's RX Descriptor. The checksum covers the frame data fields covering the entire TCP portion of the packet. If the packet is unparsable the Header Parser will provide the checksum start offset value of 34 (14 bytes Ethernet Header + 20 bytes IP header) as a default. The host must determine if the receive checksum is relevant for each frame's upper layer protocol, and compensate for the header bytes and the even byte alignment, if necessary. Receive frame checksum is always computed by Cassini, and is optional in the sense that the host can selectively ignore it whenever it does not need it, or can compute it faster in software.

Transmit frame checksum generation affects the frame format on the cable, therefore the checksum coverage and placement within the frame are rigidly defined for a given upper layer protocol. These two are supplied by the host, on a frame by frame basis, as parameters in the TX descriptor. Cassini restricts "Checksum Start Offset" and "Checksum Stuff Offset" to 16-bit alignment (the lsb of these parameters is forced to zero). Transmit checksum is optional in the sense that it will not be inserted by Cassini unless specifically enabled in the TX Descriptor of the frame.

6.1.3 Receive Completion Ring Descriptor

Each packet written into host memory is followed by a post of a descriptor on the RX Completion Ring as governed by the HRP control logic. (Some cases actually result in two posts as described later in this document.) Each RX Completion Ring descriptor is 32 bytes and has the following composition (the fields are fully defined in the Cassini Software Overview chapter):

Word 1:

- [63:62] - Descriptor Type
  a. 00 - owned by H/W
  b. 01 - release of stale flow buffer
  c. 10 - release of non-reassembled packet
  d. 11 - release of flow packet
- [61] - release current header buffer
- [60] - release current data buffer
- [59] - release current flow
- [58] - split packet
- [57] - release next data buffer
- [56:55] - skip_num_desc
- [54:41] - current data buffer index
- [40:27] - current packet data offset
- [26:13] - current packet data size
- [12:0] - reserved

Word 2:

- [63:50] - current header buffer index
- [49:44] - current packet header offset
- [43:36] - current packet header size
- [35:22] - next data buffer index
- [34:0] - reserved

Word 3:
- [63:48] - SAP
- [47:42] - L3 header offset
- [41] - encrypted packet
- [40:35] - load balancing CPU number
- [34:30] - reserved
- [29] - no assist
- [28] - force flow lookahead
- [27:25] - opcode
- [24:19] - flow ID
- [18:12] - checksum start offset
- [2] - jumbo header split enable
- [1] - jumbo_pkt
- [0] - small_pkt

Word 4:
- [63] - abort
- [62] - bad packet
- [61] - reserved
- [60] - hash filter pass
- [59:44] - address hash value
- [43] - descriptor valid bit (set by h/w, cleared initially by s/w)
- [33:30] - perfect address match
- [29:16] - packet length
- [15:0] - TCP checksum value

6.1.3.1 Values Written into RX Completion Ring Entries by Hardware

The RX Completion Ring Descriptor entries are 32 bytes in length and consist of four 64 bit entries labeled Word 1 through Word 4. The fields have been listed above and are defined in detail in the software and hardware architectural documents. Whereas most fields are straight forward with regards to what hardware should place into them, there are other fields whose values are context dependent. For the system to work properly the driver and the hardware must have a common understanding and interpretation of what hardware will place into the fields at any given time. This section outlines what hardware will write into the descriptor fields.

Word 1

The type and split packet field definitions are implementable unambiguously. Hardware will handle the other fields as described below:

- descriptor type [63:62] : indicates type of packet released to driver. See Cassini Software Overview for a bit description.

- release_header[61] : H/W will set this bit only if this descriptor write is for a packet that has a header buffer component which was placed into the last 256 byte cell of the page sized buffer.
- release_data[60] : H/W sets this bit only if it has completed its use of the data buffer specified in the buffer index. S/W is free to recycle this buffer or even invalidate the page as H/W will make no further accesses to it.
- release_flow[59] : H/W sets this bit if it has determined the flow should be torn down due to various reasons including end of stream, TCP control flags set, packet batching detects no further stream packets, etc. Jumbo packets and non-reassembly packets always have this bit set.
- split_packet[58] : Indicates this packet spanned two reassembly buffers, and that the next_index field is valid. The driver will process a split buffer when this bit is set.
- release_both_data[57] : H/W sets this bit only if it has completed its use of the data buffer specified in the buffer index and the data buffer specified in the next_index. S/W is free to recycle these buffers or even invalidate the page as H/W will make no further accesses to it.
- skip_num_desc[56:55] : Indicates the number of descriptors after this one which the driver should skip. They are No Ops in order to fill the cacheline.
- data_index[54:41] : H/W will always write the Reassembly Buffer Index value from the Flow Reassembly DMA Table into this field if doing reassembly and the No-Reassembly Buffer Index from the No-Reassembly DMA Register if not doing reassembly.
- data_offset[40:27] : This field will get 0 unless reassembly is taking place with this packet. When reassembling, the location to the beginning of the packet in the page goes into the field. This assumes all reassembly buffers will always be page aligned. (The number of leading zeroes will vary if the page size is not 8KB.)
- data_size[26:13] : The value in Rx_ctrl_fifo[16:3] goes into the field unless there is no data buffer affiliated with this packet (i.e. only header buffer used) which will cause 0 to be written instead.

Word 2

The context dependent upper bits are formed as follows:

- header_index[63:50] : The content of Headers Buffer Index field of the Headers DMA Register will always be written into this entry.
- header_offset[49:44] : The value Headers_DMA_Register[27:22] goes into this field. (Note that if the page size is not 8KB this value as well as the descriptor field width need modifications.)
- header_size[43:35] : If there is no header buffer for the packet will write 0 into this field. Otherwise, will write (Rx_ctrl_fifo[18:12] - 1).
- next_index[34:21] : Either the index of the data buffer which holds the overflow from the payload of the packet being reassembled (reassembly in progress) or a don’t care value if no reassembly in progress.

The least significant 29 bits of Word 2 is easy for hardware to form:

- Word 2[28:0] <- Rx Control Fifo[28:0]

Words 3 & 4

See description in Cassini Software Overview, Receive Completion bit descriptions.

6.1.3.2 RX Completion Entry Write Batching

To prevent read-modify-write operations on the host memory bus from writing completion entries back that fill only a partial cacheline, multiple entries can be concatenated and written together to fill out a cacheline. The grouping can be done when logic detects more packets in the RX Fifo following the current one. Should no subsequent packets be in the fifo and a cacheline only be partially filled by writing the current entry or entries to the completion ring, hardware can fill the cacheline with garbage/zeroes and set the Num_desc_skip field within Word 2. The Skip field is normally set to zero indicating that the next descriptor will immediately follow the current one. Should there be a gap between them due to cacheline filling optimizations, the field indicates in units of 32 bytes / single descriptor entries the number of “null” entries between the current descriptor and the next active one.

6.1.3.3 Flow Reassembly DMA Controller

The meat of the HRP logic is the controller which directs packets stored in the RX Fifo into host memory buffers and updates the RX Completion Ring. This controller and surrounding logic modules will be described in this section of the document. Note that the algorithm that is shown in flow chart fashion is not intended to represent exact state and timing level implementation detail. The purpose is to describe in reasonable detail the actions the controller takes without walking through the actual Verilog code.

The overall processing for each opcode was described in the section above. The behaviour of the HRP controller for each opcode is described in this section.

6.1.3.4 Controller Entry Point

Shown in Figure 6-9 on the left, is the basic wait loop the controller idles in until the Rx Control Fifo has an entry signifying the presence of a packet in the Rx Data Fifo to be transferred to host memory. Shown on the right side is the processing flow the controller does for non-TCP packets or packets unaffiliated with a TCP stream.

Figure 6-8 HRP Algorithm - Starting Block

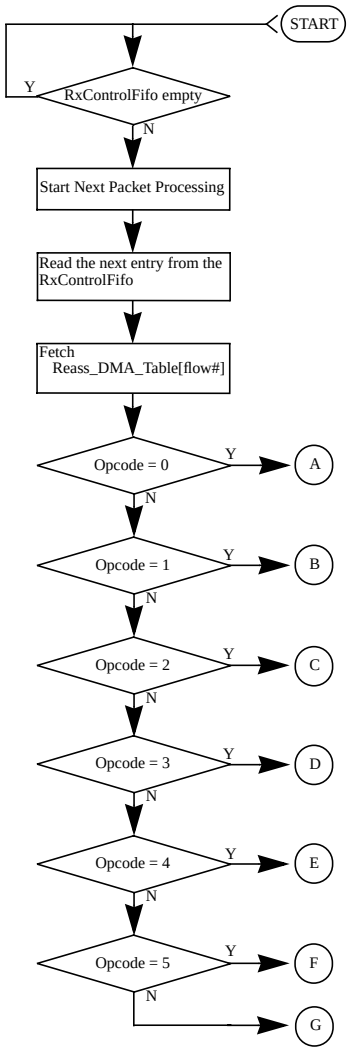
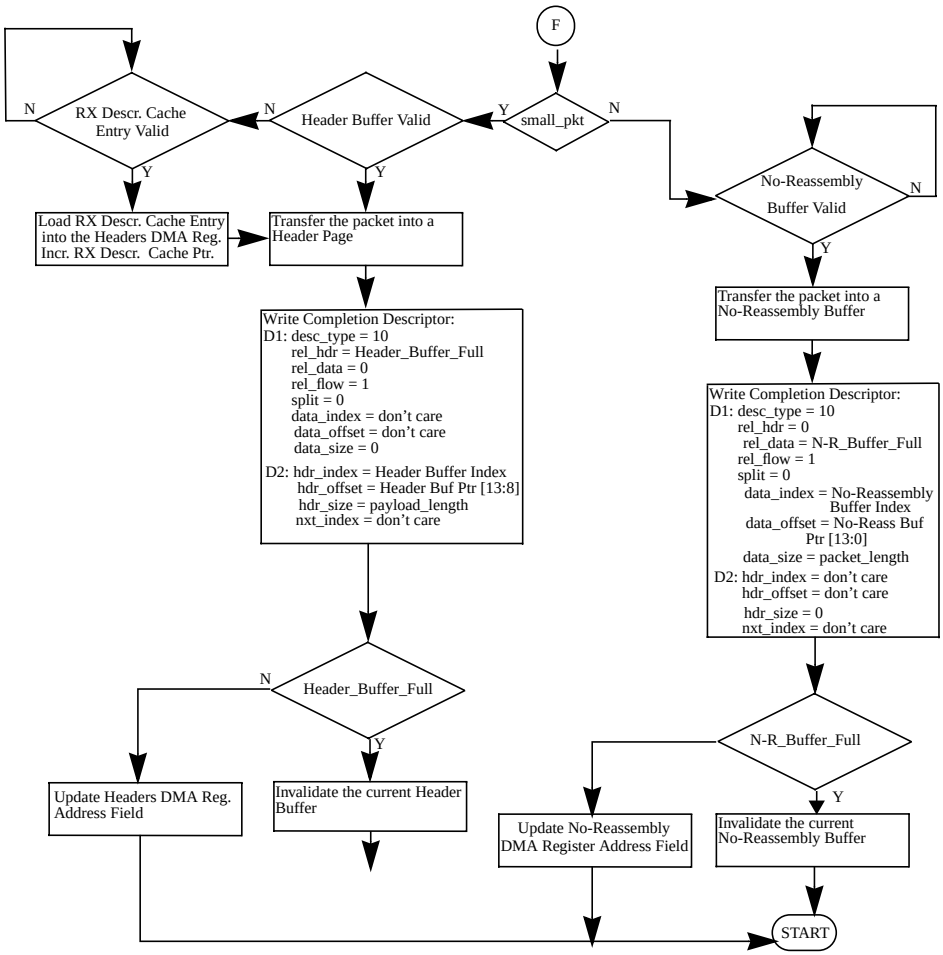


Figure 6-9 HRP Algorithm - Part I



From the flow chart on the left, once an entry is in the RX Control Fifo, it is read from the fifo to inspect the information fields. The “Start Next Packet Processing” signal is sent to the packet batching controller described later. The Flow Reassembly DMA Table at the Flow ID entry is fetched to obtain the status of a TCP stream reassembly potentially in progress. This table’s fields and use will be described later. Finally, the opcode from the Rx Control Fifo is inspected and processing in the state machine branches accordingly.

6.1.3.5 DMA State Registers and Flow Reassembly DMA Table

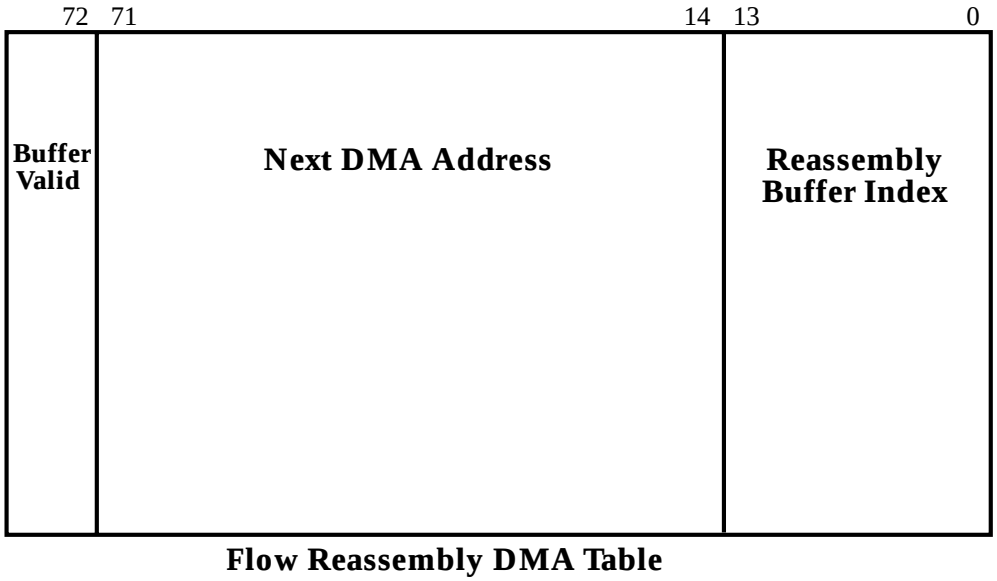
The top half of Figure 6-6 shows the storage mechanisms for RX host memory buffer address pointers maintained locally on chip. There are two 79 bit registers which hold the 64 bit address of the No-Reassembly DMA buffer in active use and the 64 bit address of the page size buffer used for small packets in active use. Both registers have a bit to indicate if the entry is valid and bits to hold the buffer index which must be written into the completion ring descriptor. The registers are loaded from the RX Descriptor Cache. Figure 6-10 below shows the format of the registers and Flow Reassembly DMA Table.

The address field of the No-Reassembly DMA Register is initially loaded with the sum of the descriptor cache address field plus the MTU_offset from the Page Control Register. (Note that the MTU_offset field can be combined with bits [7:5] of the address descriptor with a simple function rather than added.) Also, the value from the Num_MTU_buffer field of the Page Control Register is loaded into a holding register. Each packet transferred into the no-reassembly buffer causes the initial value for Num_MTU_buffers to be decremented. Once the holding register reaches zero, no more non-reassembly packets will be packed into this buffer and the buffer is returned to the driver. Also, the initial value placed into the address field of the No-Reassembly DMA Register is increased by the value in the MTU_buffer_stride field of the Page Control Register after each packet transfer and points to where the next non-reassembly packet will be written into the buffer.

The address field of the Headers DMA Register is initially loaded with the value from the cache entry but is advanced by 256 after each packet has been written into host memory. Thus, it serves as a pointer to the beginning of the next 256 byte cell within the buffer. Similar to the non-reassembly case described above, the Page Control Register holds a value (Num_Header_buffers) which is loaded into a holding register and decremented per packet transferred into a header buffer. Once it reaches zero, the header buffer is invalidated.

An asynchronous process not shown in the flow diagrams strives to keep the No-Reassembly and Header DMA Registers valid by loading valid entries into them from the RX Descriptor Cache.

Figure 6-10 DMA State Registers and Flow Reassembly DMA Table



Flow Reassembly DMA Table

Headers DMA Register

No-Reassembly DMA Register

The Flow Reassembly DMA Table entries are similar to the DMA state register entries. The table is comprised of 64 entries 79 bits wide. The buffer index field is used the same as for the DMA registers; the value is written into the completion ring descriptor. The valid bit indicates if the HRP has a flow established at the Flow ID equal to the table index. The address field is titled “Next DMA Address” and is initially loaded with the value from the descriptor cache. It is advanced by the number of bytes transferred into the reassembly buffer in host memory. When the next packet of the TCP stream comes along, the value stored in the address field indicates where DMA needs to commence to concatenate the payload of the current packet with that of the previous.

6.1.3.6 Non-TCP Packet or Non-Reassembleable TCP Packet (Opcode 5)

The right side of Figure 6-6 outlines the controller's activities for this case. For packets 256 - swivel_offset bytes or smaller, the packet will be packed into a page size buffer allocated for headers. Should such a buffer not be currently available, the controller waits until the RX Descriptor Cache provides one. The completion ring is updated and if this packet fills the last 256 byte cell of the header buffer the next descriptor is read from the descriptor cache into the Header DMA Register and the valid bit set overwriting the current contents. Should the Descriptor Cache not have an entry yet, the valid bit for the register is cleared until the register gets loaded from the descriptor cache.

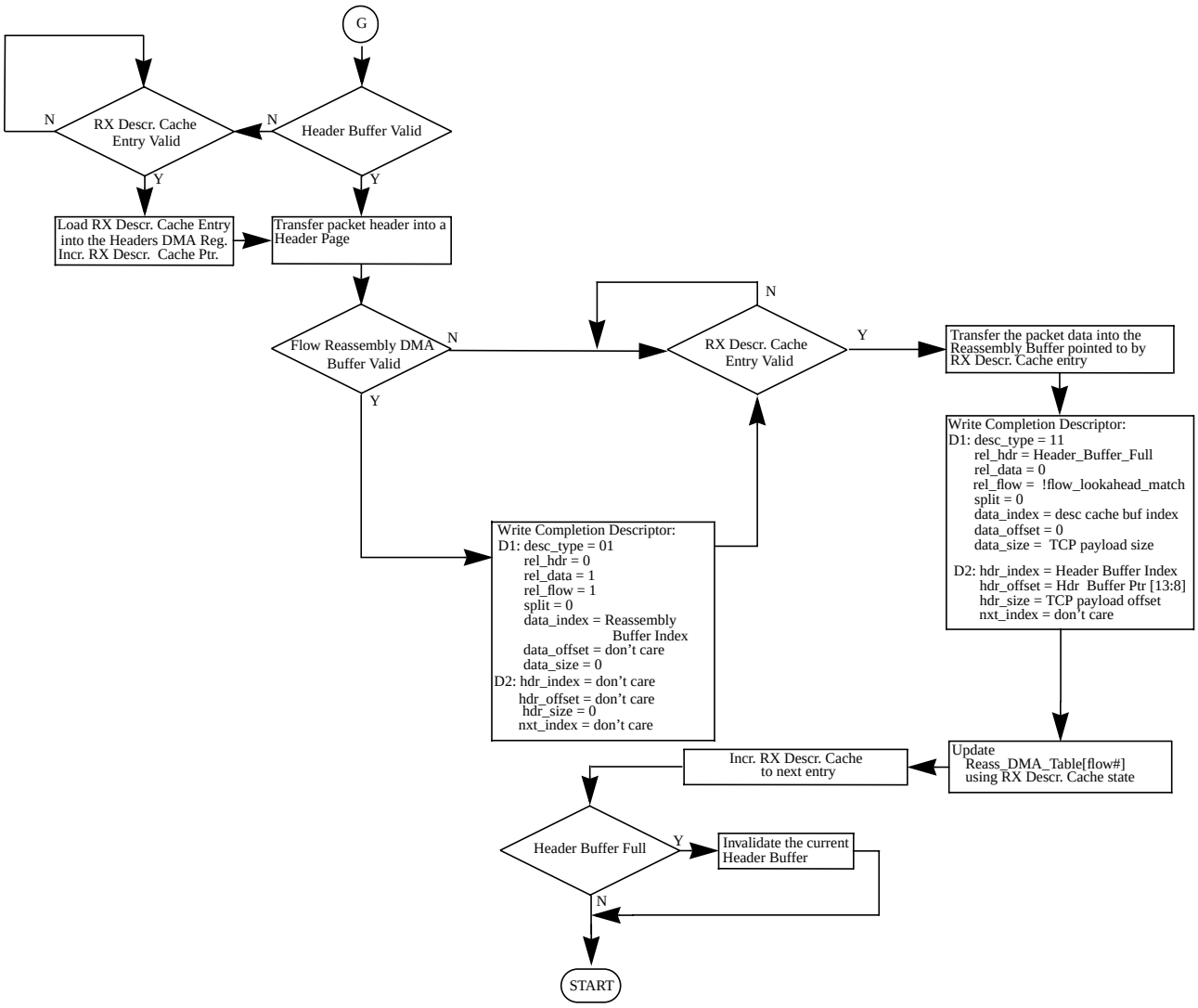
Should the packet be larger than 256 - swivel_offset bytes, the processing is similar except that a valid no-reassembly buffer needs to be available in the No-Reassembly DMA Register. If not, the controller waits until one is obtained from the RX descriptor cache. Once the packet is transferred to memory and the completion ring updated, the No-Reassembly DMA Register needs to be loaded with the next entry from the RX Descriptor Cache which if empty causes the No-Reassembly DMA Register's valid bit to be cleared.

6.1.3.7 First Packet of New Flow (Opcode 6)

This case is diagrammed in Figure 6-11. Note that in this and other cases to be described when there is no header buffer valid a check is made to see if the RX Descriptor Cache has a valid entry which holds a free buffer descriptor. If there is, the buffer pointer from the cache is loaded into the Headers DMA Register and the pointer within the cache is incremented to the point to the next cache entry. Similarly, when there is no valid entry in the Flow Reassembly DMA Table when a packet is to be reassembled, the RX Descriptor Cache is checked for a valid entry. If none is present, the state machine waits until descriptors are loaded into the cache. The address from a valid entry will become the pointer to the beginning of a reassembly buffer and the packet payload is copied into it. Later, the state machine updates the DMA Table with the address increased by the payload size.

Figure 6-11 diagrams the opcode 7 case also. What distinguishes opcode 6 is that there is no valid Flow Reassembly DMA buffer valid at the flow ID entry of the Flow Reassembly DMA Table.

Figure 6-11 HRP Algorithm Part II



6.1.3.8 First TCP Packet of New Flow with Database Full (Opcode 7)

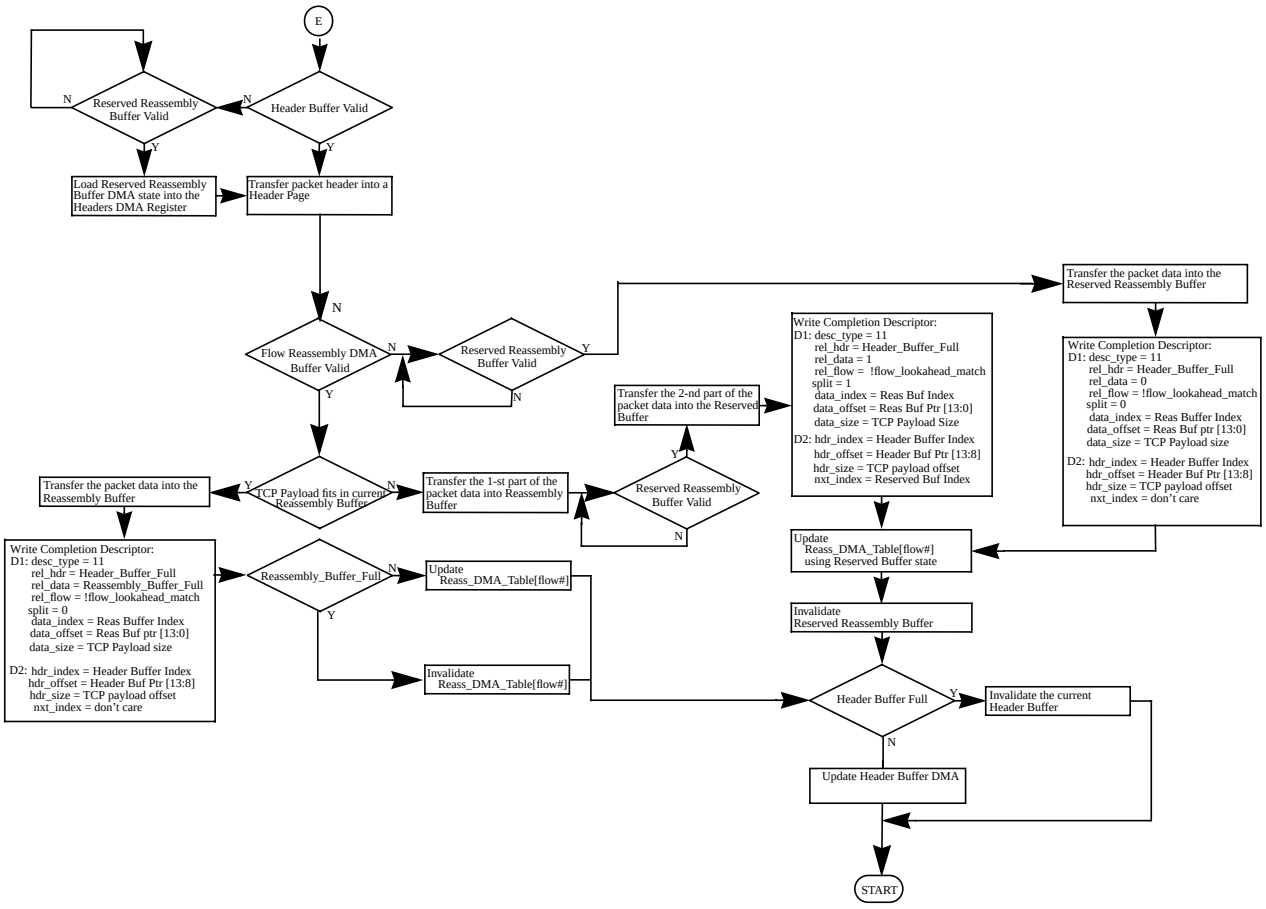
This case is also diagrammed in Figure 6-11. The HRP infers that the database is full by receiving opcode 7 (which is actually conclusive evidence) and by detecting that there is a currently valid Flow Reassembly DMA buffer valid at the flow ID passed to the controller and an active flow present their also. The HRP releases the old flow before transferring the current packet to the host memory reassembly buffer.

6.1.3.9 TCP Packet in Middle of Stream (Opcode 4)

This case is diagrammed in Figure 6-12. Two things may be worthwhile explaining in the algorithm. Close to the entry point a test is made to see if there is a valid reassembly buffer present. Opcode 4 occurs after previous reassembly has taken place into a reassembly buffer which generally implies a valid buffer at least was present in the past. What might make a valid buffer no longer present would be if the previous packet's payload exactly filled to the end of the page size reassembly buffer. That would cause the buffer to be released to the driver and an interval potentially exists where there is no new reassembly buffer given to the hardware yet by the driver or the hardware had yet to fetch it. The second point of explanation is affiliated with this. The flow active test made early in the algorithm will fail if the previous packet's payload exactly filled the reassembly buffer. (Flow Active, to be true, indicates at least one byte has been copied from the Rx Data Fifo into the reassembly buffer whose address is in the address field of the Reassembly Table.)

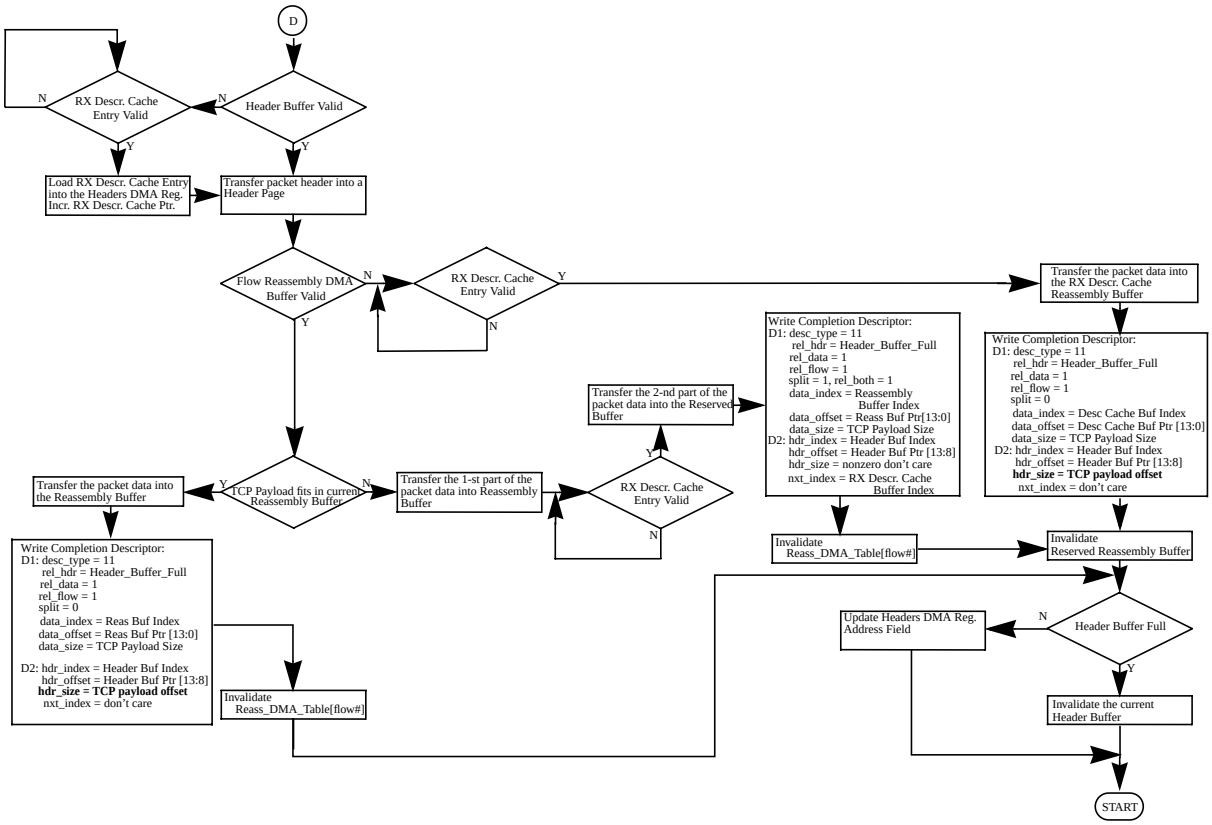
Note the algorithm covers the situation where the payload of the packet spills over into a second reassembly buffer (see center of diagram). The completion ring descriptor posted will have the split packet bit set and will include the buffer indexes of the first buffer (data_index field) and the second (next_index field).

Figure 6-12 HRP Algorithm Part III



6.1.3.10 Last TCP Packet in the Flow (Opcode 3)

Figure 6-13 HRP Algorithm Part IV

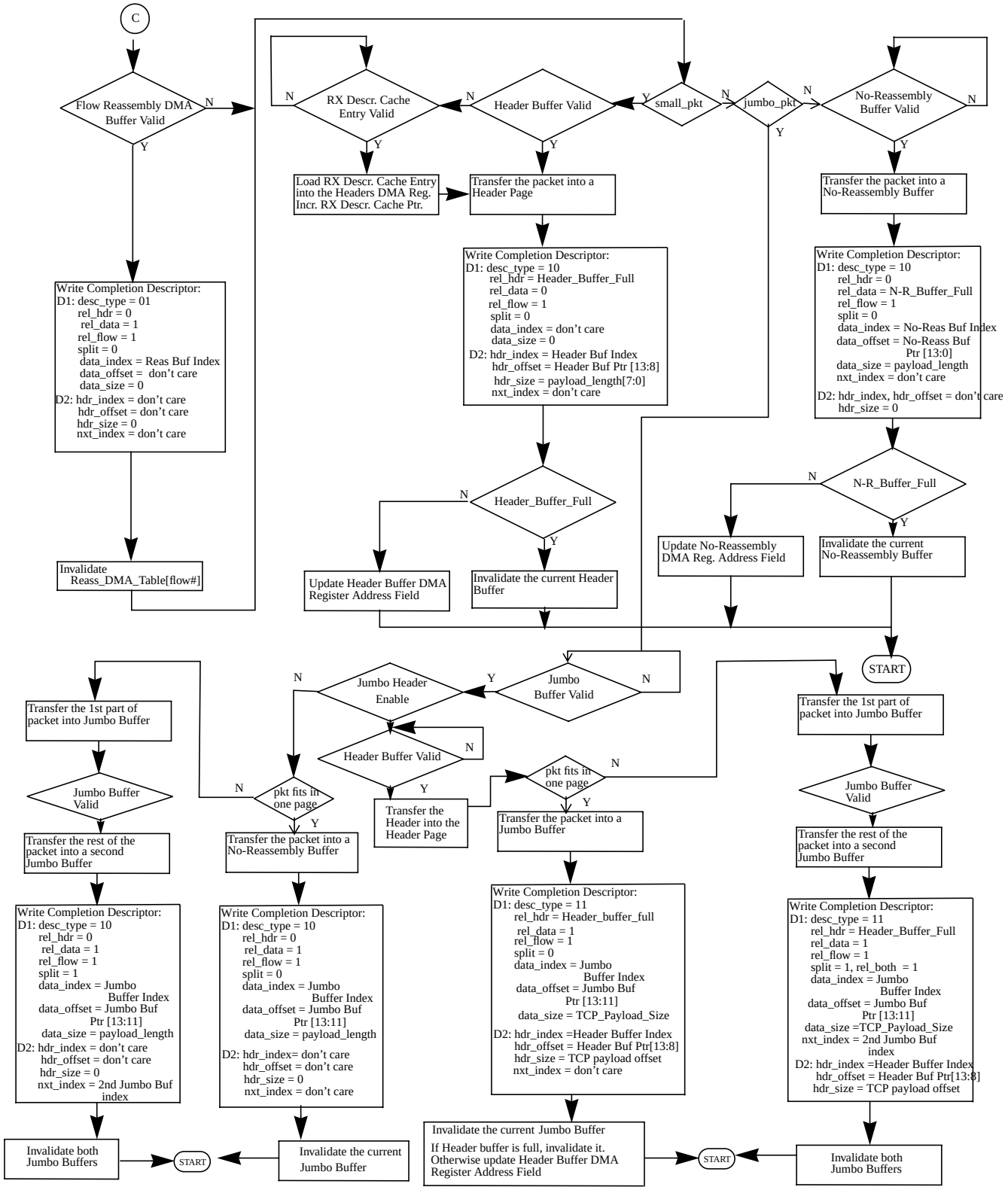


This is a case rather similar to opcode 4 and is diagrammed in Figure 6-13.

6.1.3.11 TCP Packet with Out of Order Sequence Number (Opcode 2)

This case is diagrammed in Figure 6-14.

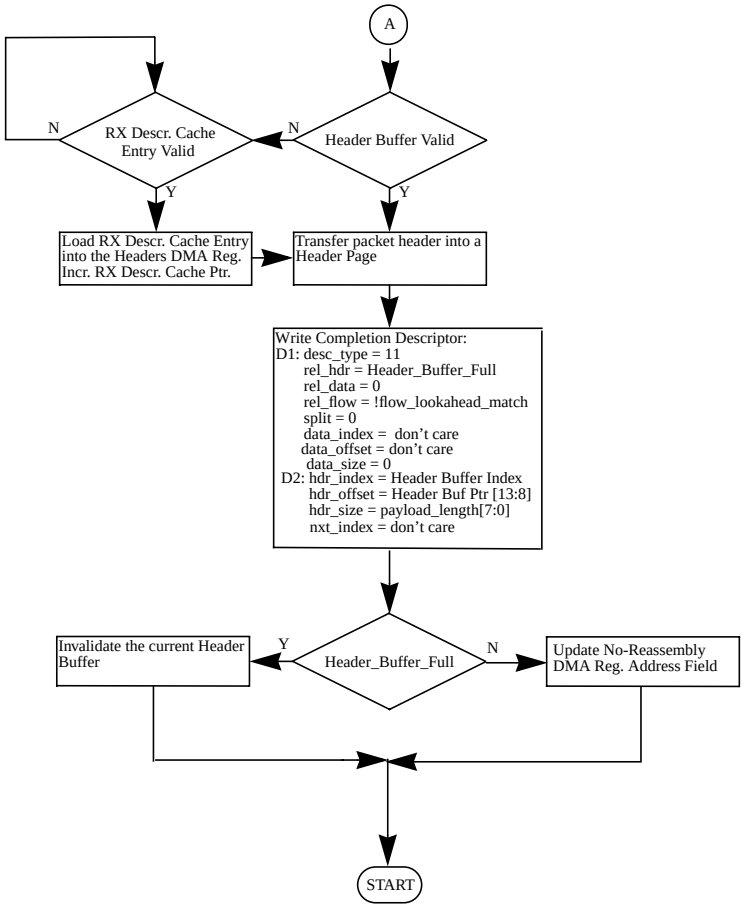
Figure 6-14 HRP Algorithm Part V



6.1.3.12 TCP Control Packet (Opcode 0)

The case of a TCP control packet with no flags set is in Figure 6-15.

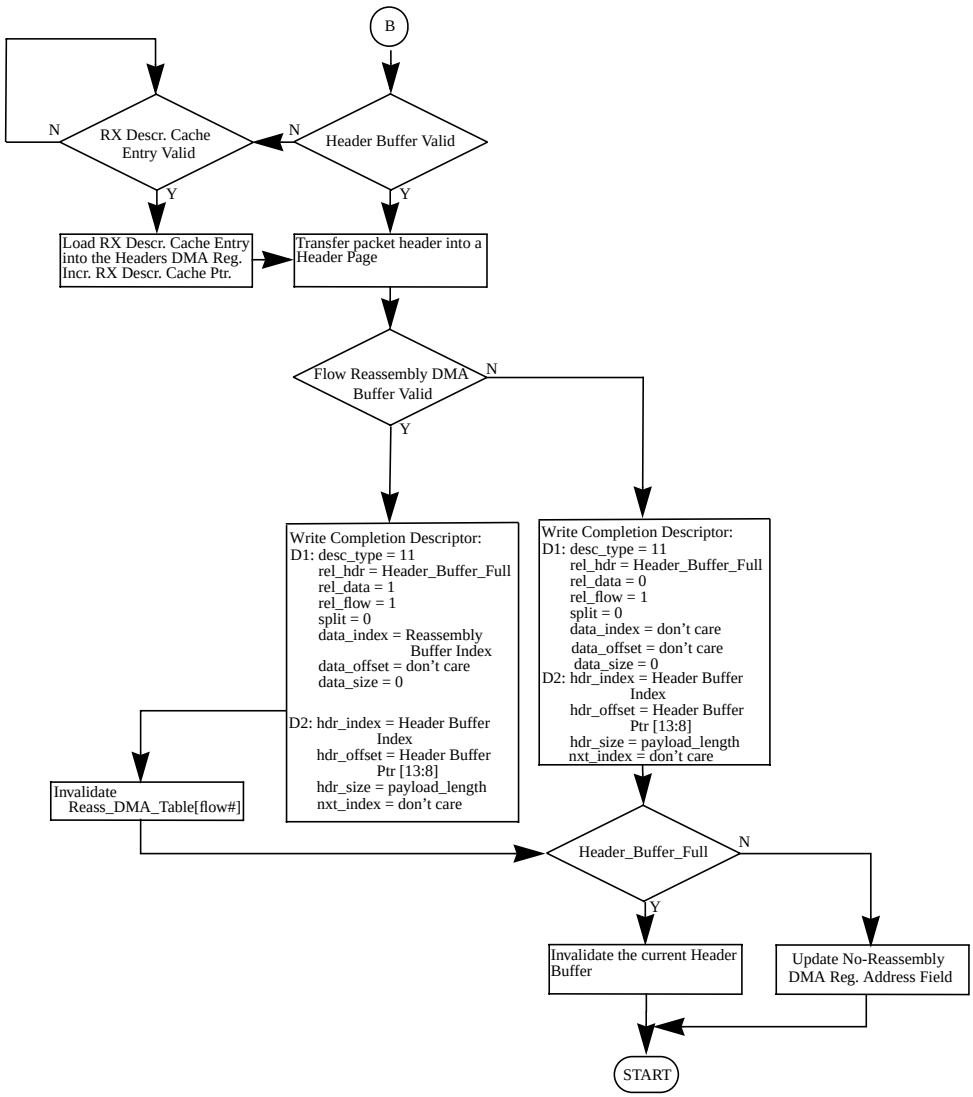
Figure 6-15 HRP Algorithm Part VI



6.1.3.13 TCP Control Packet with Flags Set (Opcode 1)

This is the final opcode case and diagrammed in Figure 6-16.

Figure 6-16 HRP Algorithm Part VII



6.1.3.14 Dynamic Packet Batching

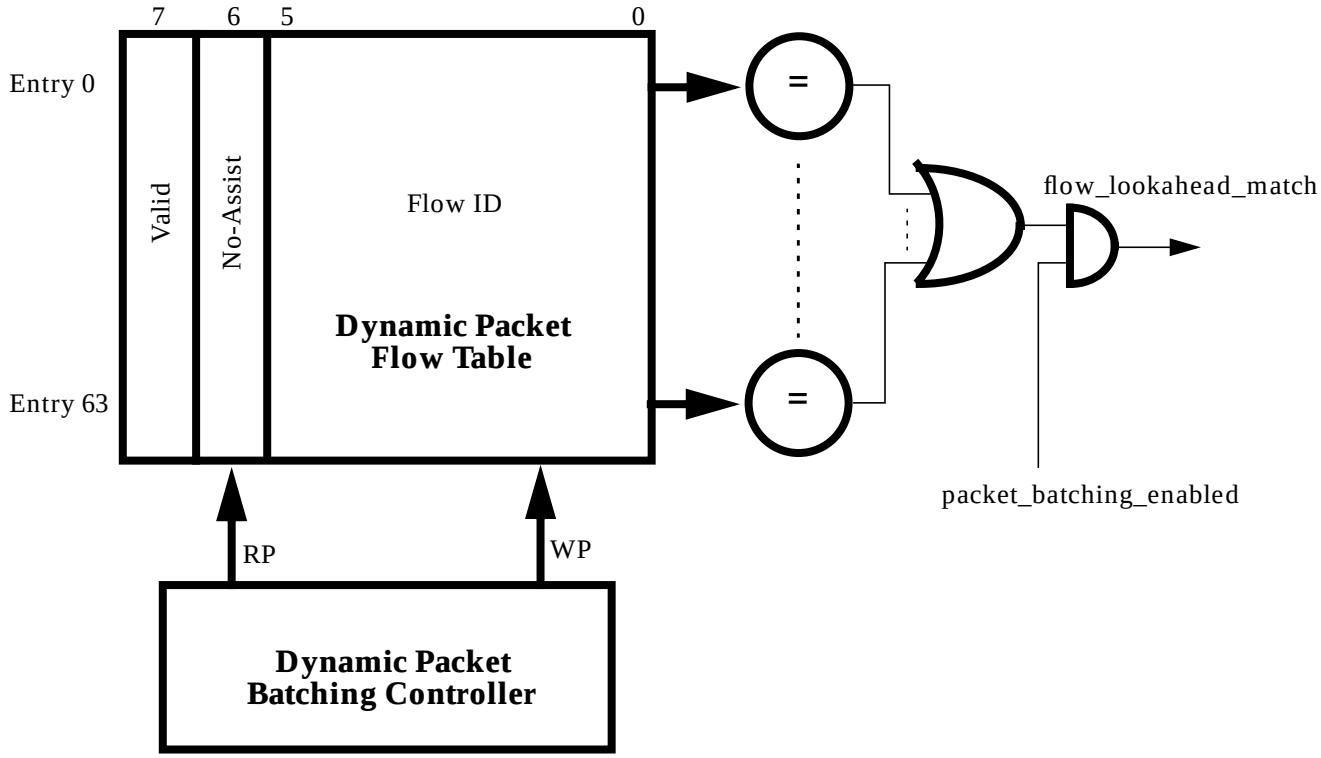
To reduce packet header processing overhead, packet batching logic indicates to the driver if packets appear to be closely spaced and within the same TCP stream or flow. The protocol stack, if optimized for this, can reduce packet header processing overhead.

Logic Overview

Logic in the HRP block scans a table to see if the packet whose TCP payload is being written into a reassembly buffer is being followed by more TCP packets of that same flow. If so, the "flow_lookahead_match" signal referenced in the

HRP algorithm charts is asserted and the descriptor written into the RX Completion Ring does not set the Release Flow bit (Word 1 - bit 59). When the flow terminates due to end of TCP stream or some condition resulting in no further reassembly, the Release Flow bit gets set, of course. When dynamic packet batching is enabled, the bit gets set prior to the end of the stream if no packets are found in the Dynamic Batching Flow Table affiliated with the flow. The table holds 64 entries with one entry per packet. The basic table and affiliated logic is shown in Figure 6-17.

Figure 6-17 Dynamic Packet Batching Manager



An entry is written into the table when the Rx Control Fifo is updated with a new packet if there is space in the table. Once the table is filled, new entries are made once space is created as RX packets get written into host memory. Each entry contains the six bit flow ID (if any) for the packet, a valid bit to indicate if the entry holds legitimate information, and a bit indicating if the packet is a reassembly packet (via No-assist bit). The table is implemented with flip-flops so each entry can be monitored simultaneously. The table search checks every entry against the flow ID of the packet whose payload is getting copied into a reassembly buffer. The comparison verifies the valid bit is set and the no-assist bit is clear.

Packet Batching Controller

The 64 packets following the TCP reassembly packet are checked for flow association as mentioned earlier. The Rx Control Fifo contains up to 256 packets and sufficient information that would allow this check to be made.

Given it is a dual port RAM, however, fast parallel checking of multiple entries cannot be done. Thus, the batching table is a discrete flip-flop based window of the 64 packets which follow the packet under test which allows parallel comparison on all entries.

The controller must govern addition and removal of packet entry information into the packet batching table. It is a logically a circular ring structure with pointers to track head and tail. Packets are entered when the table has an entry available (table not full) and either a new packet is about to be placed into the Rx Control Fifo or previous packets which could not fit into the table but were placed previously into the Rx Control Fifo need to be entered. Packets get removed from the table when the HRP controller begins its work. The signal “Start New Packet Processing” is generated at the beginning of that controller’s work and can be used.

6.2 Jumbo Frame Support

Cassini enhances system network performance through TCP stream packet reassembly which requires no change to protocol standards. There has been discussion within industry for years about support for ethernet packets larger than 1518 bytes or 1522 bytes including VLAN tag. This would decrease system overhead by reducing the number of packet headers per unit data when large streams of data are being transmitted or received. There is currently no standard for these oversized ethernet frames (jumbo frames). The concept has achieved popularity, however, and with very modest change to the Cassini architecture support readiness has been included. Naturally, if momentum towards a standard accelerates but in a direction different from current industry proposals, changes will be made as deemed necessary in the jumbo frame support logic. Even without standardization, point-to-point communications using jumbo frames is possible with equipment using the same frame structure. This type of use may become more frequent prior to standards level consensus.

As was true for most of the Cassini hardware performance features described thus far, the majority of the impact of jumbo frame support is on the RX side of the machine. These will be described along with a comment on TX considerations.

6.2.1 Ethernet Jumbo Frame Structure

Given there is no existing standard, the structure has not been finalized. Based on current discussions within the industry there is a likely format jumbo frames will take and is shown below.

It is anticipated that following the destination and source address fields, a new type signifying a jumbo frame will be placed into the length/type field. Eight bytes of LLC - SNAP will follow and after those the jumbo packet data and four byte CRC fill out the packet.

6.2.2 Cassini Hardware Support Mechanisms

The following have been incorporated into the hardware architecture to support jumbo frames:

- The IPP Fifo will be 9.2KB, sufficient in size to accommodate jumbo frames of relevant interest
- The IPP Header Parser will check for the jumbo frame ethernet type after the MAC source address field. If found it will continue to parse the packet rather than categorize it as NO_ASSIST.
- A 16 bit register (Jumbo Type) will be loaded by software with the jumbo frame ethernet type value. Given there is no standard establishing the value it is made a variable software initializes. If the system does not want Cassini to parse a header looking for a jumbo frame ethernet type, the driver should place a benign value such as 0x0000 into the register which causes the parser to process the header in its normal manner.
- When a jumbo frame is encountered, the header parser will parse it looking for the beginning of the TCP packet and generate the various datum (eg. TCP_CHKSM_START, TCP_PAYLOAD_OFFSET, etc.) that normally are generated for standard size TCP/IP packets which meet the parsing criteria. If the jumbo packet does not meet TCP packet reassembly guidelines, the NO_ASSIST flag is set as is normal for this situation.
- Additional logic within the IPP block will generate a signal indicating a jumbo packet has been received. This signal results from comparing the packet size reported by the RX MAC with the value software has written into a 14 bit register (Jumbo Minimum Size). If the incoming packet length is at least that of the value held in the register, the jumbo frame received (JFR) signal is asserted.
- Jumbo packets which meet the normal Cassini TCP flow reassembly rules can have their headers split from the TCP payload if this mode is enabled via a register (Jumbo Hdr Split Enable) programmed by software.
- The Flow Database Manager (FDBM) searches for matches in its database when incoming jumbo frame packets arrive as it does for non-jumbo frame packets with slight modifications to its normal algorithm. Should a match be found, the flow affiliated with that match should be torn down. This results in an Opcode 2 being generated to the HRP. Should no match be found, none will be established for this packet even if it met conditions under which a flow would have been setup for normal length packets. An Opcode 5 is generated to the HRP. Note that a jumbo frame will only result in either an Opcode 2 or 5 generated to the HRP.
- The HRP expands the Opcode 2 and 5 cases for jumbo frame support. Inputs to the HRP include the Opcode, JRF (jumbo frame received), and jumbo frame header split enable (JF_hdrsplit_enb).

6.2.3 RX Jumbo Frame Packet Data Structures

The Cassini RX packet data structures consist of a single RX completion ring whose entries relay information about the packet received from the network and a single RX free buffer ring which contains pointers to page size buffers. A

buffer from the free buffer ring is allocated on a demand basis to be either a header buffer (contains headers from reassembly packets or packets 256 bytes or smaller), an MTU buffer (packed 1500 byte buffers), or a reassembly buffer. A fourth page size buffer category, jumbo frame buffer, is added to this set. Software should use a page size so that any jumbo frame received would not overflow two pages. If header is split for example, the data portion would not exceed a size of two pages. This includes the crc. If header is not split remember that the mtu buffer offset and swiveling is applied, which take up room in the page.

6.2.3.1 Non-Split Jumbo Frame Packets

The packet containing the header plus payload is DMA'd into one or more jumbo frame buffers depending upon its size. The beginning of the frame is offset from the start of the buffer by the amount specified in the MTU_offset field of the Page Control Register so the driver can prepend network information affiliated with the packet. The non-split jumbo frames will be swiveled according to the programmable swivel offset and have the mtu buffer offset prepended to the packet. Checksum is done to the end of the packet just like non-reassembly packets. In addition to the standard items such as buffer indexes, the RX completion ring entry will specify:

- release of non-reassembled packet
- release of current data buffer
- release of current flow
- no-assist bit set
- packet header size of zero
- should the packet be larger than a page and require two pages, the split packet bit and the release next data buffer bit are set.

6.2.3.2 Header Split Jumbo Frame packets

If the jumbo packet meets flow reassembly rules and jumbo header splitting is enabled, the HRP places the header into a header buffer and the TCP payload into a jumbo frame buffer. There is no swivel or MTU_offset applied to the starting address of the TCP payload DMA. Checksum is done to the end of the packet just like non-reassembly packets. The completion ring entry is updated similarly to the non-split case above but the no-assist bit is clear and the packet header size field reflects the IP and TCP header lengths.

6.2.3.3 Batching without Reassembly

The purpose of batching without reassembly is to allow batching to occur if reassembly is not going to be supported in Solaris. Also it would allow batching to occur in streams for which reassembly does not make sense (http streams which don't pass a lot of data). The HRP supports batching by setting or not setting the release_flow bit in the completion descriptor. It tells software if there are any more packets of the same flow in the Rx fifo forthcoming. This typically only occurs for reassembly flows since their flow_ids are valid. To support Batching without Reassembly, the microcode would be written so that although flows would be set up, some or all of these flows would generate the opcode 2 with the force_fla bit set in the HP/IPP interface. The force_fla flag

would indicate to the IPP that the flow_id of this packet should be considered in the flow matching that takes place when a descriptor is written. The opcode of 2 would instruct HRP not to reassemble the data. For IPP, the change from typical behavior would be to validate the flow_id in the CAM (for matching) although the opcode is a non-reassembly opcode. The change from typical behavior for HRP is that a match is done on the currect flow_id with all the valid flow_id's in the CAM with opcode 2. The change from typical behavior for the FDBM is that it forces an opcode of 2 to the IPP when the force_fla bit is set.

6.2.4 TX Jumbo Frame Support

There are no changes that have been identified necessary to the intrinsic structure of the TX design. The TX fifo load logic must check for sufficient space before initiating a load from host memory of a packet into the TX fifo. This must be done even for the standard packet size case but now space for a larger packet must be accounted for. The DMA should break the load of a packet too large to fit the current TX fifo unused space into smaller pieces which fit as space avails.

6.3 TX and RX DMA FIFOs

The TX and RX FIFOs are implemented as RAM arrays, where both sides of each FIFO are entirely under their respective DMA block control. Given the role these FIFOs play in Cassini's functionality, along with the size and speed demands placed on these blocks, it is important to provide them with flexible diagnostics mechanisms.

Every FIFO location is accessible using PIO instructions. The access is based on a loadable pointer register, and a set of Data Registers. The FIFO read / write granularity is in 65 bits entries (64 data bits plus the tag bit).

6.4 2 Cassini+ Rx Features

To improve system level performance, Cassini+ allows applications to use all 4 PCI's interrupts.

To assist crypto sub-system, Cassini+ Cassini+ detects IPsec's AH and ESP types of encrypted packets, stores those encrypted packets into the system memory under the rx_free_descriptor_ring_2's Control.

This section describes all the rx changes for Cassini+ Cassini+ .

6.4.1 4 Rx_completion_rings/PCI interrupts

To use 3 addition PCI's interrupts, 3 more rx_completion_rings are added.

Structually, these 3 rx_completion_rings are simply the duplications of the original rx_completion_ring.

There are 3 more sets of PIO_registers to program these 3 rings, see the list in the Programmer's Model.

(1) Finishing a packet, the Rx uses 1 of the 4 completion_rings / interrupts

. INTA denotes the rx_completion_ring, the original Cassini uses

. INTB denotes the rx_completion_ring_2,

. INTC denotes the rx_completion_ring_3,

. INTD denotes the rx_completion_ring_4.

(2) INTB,C,D can be enable / disable individually.

(3) Which completion_ring will be used, depending upon

. the last 2 bits of the cpu_load_balancing number, and

. the 4 interrupts enable condition.

(4) Enabling 4 PCI interrupts, see BIM section of the Programmer's Model.

6.4.2 2 Rx_free_descriptor_rings

To separate encrypted packets from packet streams, a dedicated rx_free_descriptor_ring_2 is added.

Structually, the rx_free_deiscriptor_ring_2 is simply a duplication of the original rx_free_descriptor_ring.

There is 1 more set of PIO_registers to program the second rx_free_descriptor_ring, see the list in the Programmer's Model.

(1) The 2 rx_free_descriptor_rings work independently:

. the original Cassini rx_free_descriptor_ring,

under RX_PIO_KICK=0x4024's control,

handles non_encrypted packets.

. the added rx_free_descriptor_ring_2,

under RX_PIO_KICK2=0x4220's control,

handles only encrypted packets.

. the 2 rx_free_descriptor_rings can trigger each of the 4

rx_completion_rings, but 1 at a time.

(2) Once an encrypted packet is detected, the hardware picks the dedicated rx_free_descriptor_ring_2.

6.4.3 1 field added to a rx_completion_ring

In the third word of a rx_completion_ring,

bit[47:42]: "L3_header_offset" becomes 6 bits, Cassini uses bit[47:41]

bit[41]: “Encrypted_packet” if it is “1”, Cassini+ added.

6.4.4 1 field added to the Rx_configuration_reg, 0x4000

bit[19:16]: “Rx_desc_ring_size2”, nmber of entries in the rx_free_descriptor_ring_2. Default: 8k, 4’h8.

6.4.5 Process an encrypted packet

(1) Recognize an encrypted packet, only if

. in IPv4’s base header, PROTOCOL = 8’h32 or 8’h33 (i.e. ESP or AH)

. in IPv6’s base header, NEXT_HEADER = 8’h32 or 8’h33.

The “encrypted_packet” bit in a rx_completion_ring will thus be set.

(2) Store encrypted packets

. if the packet size <= (256 - swivel_offset) bytes, this will cause completion write back word3 small_pkt at bit 0 to set to 1 and this packet will be stored in a header_buffer as if it were a non-encrypted small packet.

. if the packet size > (256 - swivel_offset) bytes (small_pkt = 0) it will be stored in an MTU or reassembly buffer as if it were a non_Tcp packet. No header/payload splitting.

Under rx_free_descriptor_ring_2’s control.

6.4.6 Totol number of rx related interrupts

- Rx_done: 4
- Rx_free_descriptor_buffer_none: 2
- Rx_free_descriptor_buffer_almost_empty: 2
- Rx_completion_ring_full: 4
- Rx_completion_ring_almost_empty: 4.

6.4.7 Rx_free_descriptor Caches

There are 2 rx_free_descriptor caches, one for non-encrypted packets and the other for encrypted ones.

Each cache has 32 bytes, store up to 4 rx_free_descriptors.

To run Cassini+, the 2 caches have to be initialized although the cache for non-encrypted packets may never be used.

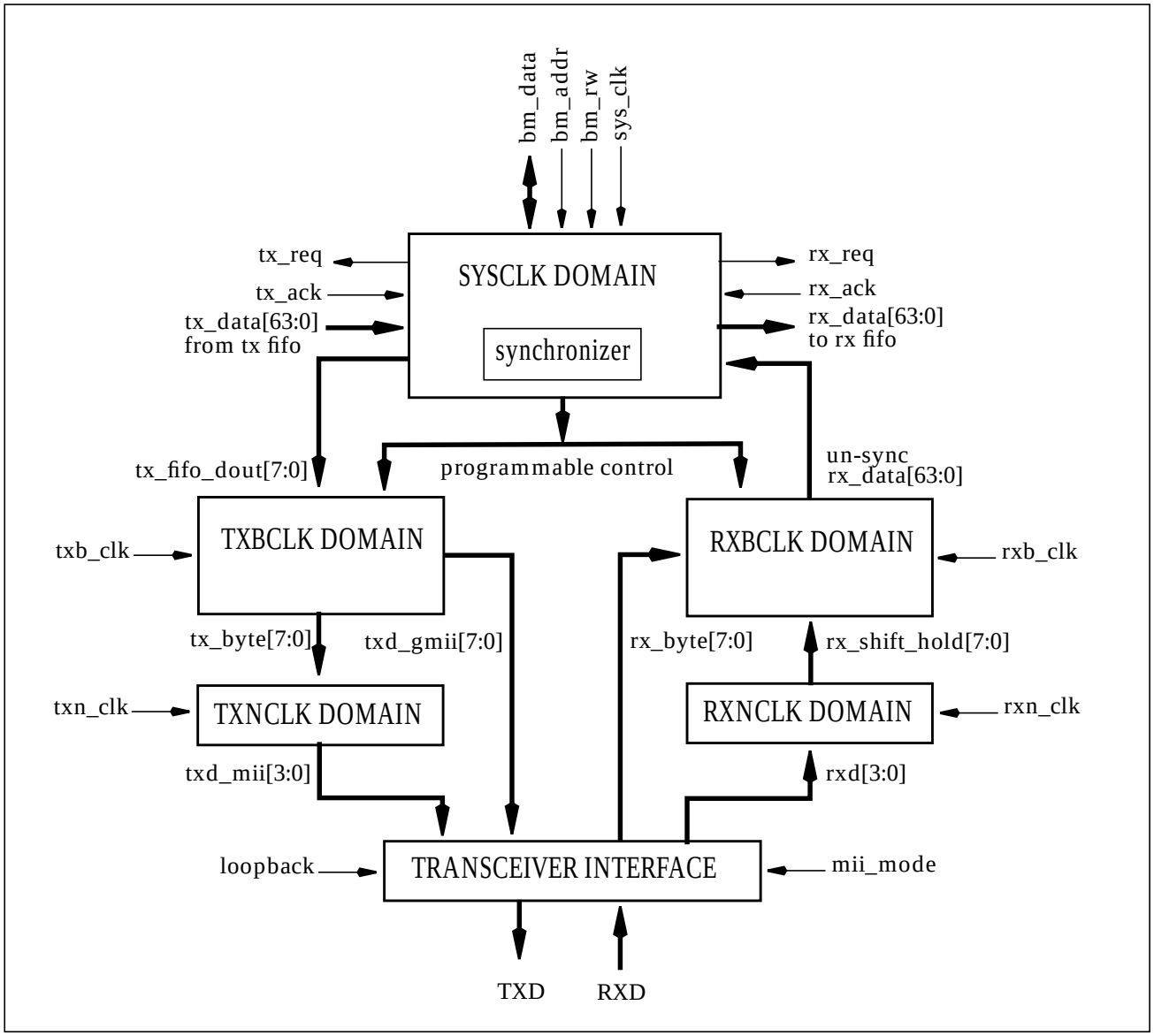
MAC, PCS, SERDES, LEDs 7

7.1 MAC blocks

The Ethernet MAC core implements the IEEE 802.3 MAC protocol for CSMA/CD networks at 10Mb/s, 100Mb/s and 1Gb/s.

There are 7 main functional blocks at the MAC top level, namely: System Clock Domain, TX Byte Clock Domain, TX Nibble Clock Domain, RX Byte Clock Domain, RX Nibble Clock Domain, Transceiver Interface and MII Management Interface.

Figure 7-1 MAC top level block diagram



7.1.1 System Clock Domain

The main task of system clock domain interface is to provide synchronized control and data signals between the BIM and the MAC core. Though the TX and RX section are also running at 125 Mhz, the synchronization is still required since MAC can operate at different clock rates in different modes.

All MAC programmable registers are located in this block. For details of these registers, please refer to the Programmer's Model chapter of this specification.

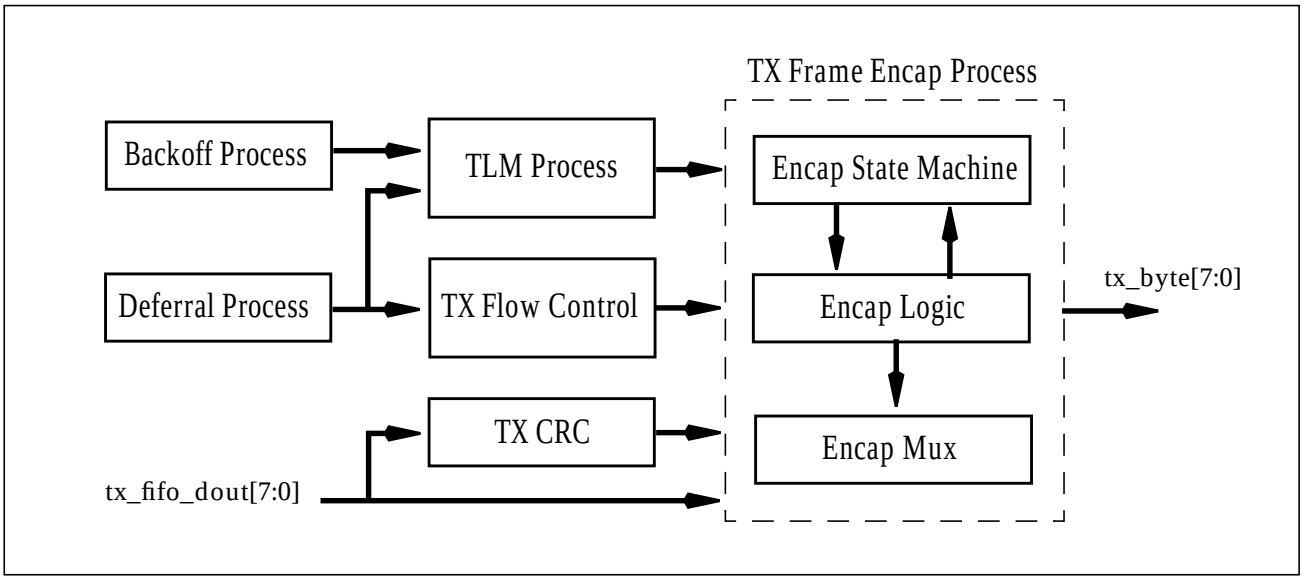
The system clock domain interface is also responsible for de-assemble 64-bit tx data from tx fifo into 8 bit tx data.

7.1.2 TXBCLK Domain

There are three sub-blocks in tx byte clock domain -- tx fifo unload, tx protocol engine, and tx GMII interface. The tx fifo unload logic provides tx fifo status to the tx state machine. The tx GMII interface provides mux select between regular tx data byte and transmit carrier extension.

The tx protocol engine is the main logic of TXBCLK domain logic. It consists of tx frame encapsulate process, deferral process, backoff process, transmit length measurement process, tx CRC generation, and tx flow control logic.

Figure 7-2 Block diagram for TX Protocol Engine



7.1.3 TXNCLK Domain

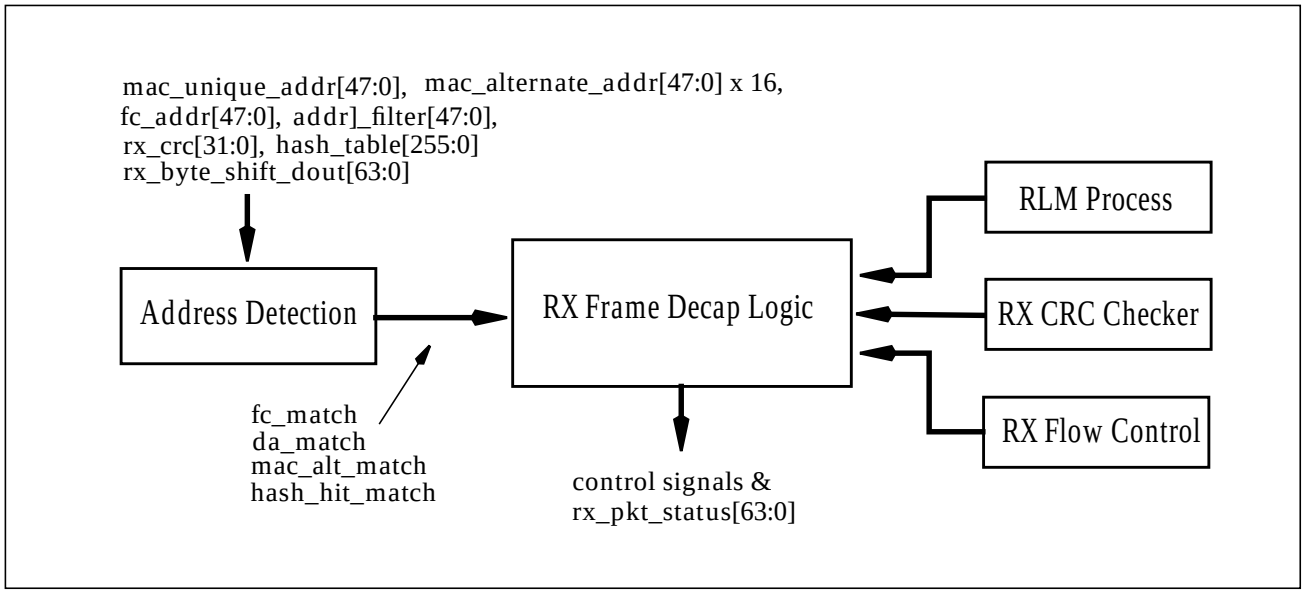
The task of tx nibble clock domain interface is quite simple. It converts one tx byte to two nibbles, and then synchronizes them to MII clock. The function of this block only apply to MII mode.

7.1.4 RXBCLK Domain

There are four sub-blocks in rx byte clock domain -- rx fifo load, rx protocol engine, rx GMII interface, and rx MII interface. The rx fifo load logic contains a 16 x 65 register file. Bytes of rx data is shifted and re-assembled into 64-bit words, which are saved in the register file and ready to be transferred to RX section. the rx GMII and rx MII interface synchronize rx_dv, rx_err, rxd[7:0], and other control signals to their own clock domain.

The rx protocol engine is the main logic of TXBCLK domain logic. It consists of rx frame decapsulate logic, rx address detection, rx CRC checker, rx length measurement process, and rx flow control logic.

Figure 7-3 Block diagram for RX Protocol Engine



7.1.5 RXNCLK Domain

The task of tx nibble clock domain interface is to re-assemble two nibbles from the media back to one rx byte, and then synchronizes them to MII clock. The function of this block only apply to MII mode.

7.1.6 Transceiver Interface (XIF)

The transceiver interface consists of muxes. Either MII or GMII signals are selected. When loopback is set true, tx data is looped back to rx data bus.

7.1.7 Transceiver Management Interface (MIF) block

The Management Interface (MIF) implements the management portion of the MII protocol. It allows the host to program and collect status information from the MII transceiver. The MIF supports three modes of operation:

"Bit-Bang" Mode

This mode of operation provides maximum flexibility with minimum hardware support for the serial communication protocol between the host and the transceivers. The actual protocol is implemented in software, and the interaction with the hardware is done via three one-bit registers: data, clock and output_enable. Each read / write operation on a transceiver register would require approximately 150 software instructions by the host.

"Frame" Mode

This mode of operation provides a much more efficient way of communication between the host and the transceivers. The serial communication protocol between the host and the transceivers is implemented in hardware, and the interaction with the software is done via one 32-bit register (Frame Register). When the software wants to execute a read/write operation on a transceiver register, all it has to do is load the Frame Register with a valid instruction ("frame"), and poll the Valid Bit for completion. The hardware will detect the instruction, serialize the data, execute the serial protocol on the MII Management Interface and set the Valid Bit to the software upon completion.

Polling Mode

As defined in IEEE 802.3, a transceiver shall implement at least one status register that will contain a defined set of essential information needed for basic network management. Since the MII does not include an interrupt line, a polling mechanism is required for detecting a status change in the transceiver. In order to reduce the software overhead, the above mentioned polling mechanism has been implemented in hardware. When this mode of operation is enabled, the MIF will continuously poll one specified register and generate a maskable interrupt when a status change is detected. Upon detection of an interrupt, the software can read a Local Status Register that will provide the latest contents of the transceiver register, and an indication which bits have changed since it was last read. This mode of operation can only be used when the MIF utilizes the "Frame Mode" for normal communication with the transceivers.

7.1.8 Hash Filtering Mechanism for Multicast Addresses

The MAC implements a hash filtering function to aid software in determining whether or not a packet with a multicast destination addresses should be accepted. The host manages a list of multicast addresses that it will respond to. For each address in the list the driver uses the hash routine to set the appropriate bit in one of the sixteen 16-bit Hash Table registers in the MAC. A device can be made a member of several groups by setting the appropriate bits in the Hash Table. In the MAC if a packet is received with the multicast bit set in the destination address, the incoming address is sent through the 802.3 32-bit CRC function. This circuit is the hash function. The high order 8 bits of the resultant 32-bit CRC is used in reverse order to select one of the 256 bit positions in the Hash Table. If the selected bit in the table is set to '1', the packet will be accepted and passed up to the host. In addition, the hardware passes up a Hash Pass bit and a Hash Value in that packet's descriptor. When the packet is received by the host and the Hash Pass bit is set, it must search the multicast address list to see if the packet's address is in the list. If it is not in the list, the packet is discarded. The Hash Value is the most significant 16 bits of the resultant 32-bit CRC value. It can aid the host in implementing a software based Hash Table of up to 64k entries.

7.2 PCS 8B/10B Block

The 8B / 10B implements the coding function, equivalent to Fiber Channel FC-1. The 10B code output of this block consists of 10-bit DC balanced Transmission Characters that reflect the state of the TxMAC and the byte stream generated by it. Special transmission characters, called Ordered sets, are generated by this block to convey frame boundary information, as well as maintaining the link during idle periods.

Similarly, this block relies on the 10B codes arriving from the far end of the link, and derives from them word alignment, frame boundary as well as the actual receive frame data bytes to be passed to the RxMAC.

This block's complies with the Fiber Channel 10-bit interface draft X3 Technical Report, allowing Cassini+ to be used with commercially available SERDES chips via its 10-bit interface pins.

7.2.1 Notation Conventions

This specification uses the FC-1 letter notation to name 10-bit Transmission Characters. Transmission Characters are named using the unencoded information byte (as presented at the GMII, for example).

Transmission Characters are represented as either K**xx.y** or D**xx.y**, where K is used for Special Codes and D is used for valid Data bytes. The **xx** field corresponds to the decimal value of the five least significant bits of the information byte, while **y** corresponds to the value of the three most significant bit. For example a preamble pattern consists of the data byte 0x55, which in FC-1 letter notation would be D21.2 (five lsb = 10101 = 21 decimal, and three msb=010 is the suffix 2).

Note that the letter notation does not define the 10-bit encoding itself. The 10-bit encoding is given in tabular form and includes one version of the character encoding for positive and one version for negative Running Disparity (both versions are identical for neutral encodings).

7.2.2 8B/10B Codes

The Valid Data Character codes are identical to those defined by FC-1, Table 22, in FC-PH Rev 4.3.

Valid Special Characters are also defined as per FC-1. Their codes are:

Table 7-1 Valid Special Characters

Special Code	Current RD -	Current RD+
	abcdei fghj	abcdei fghj
K28.0	001111 0100	110000 1011
K28.1	001111 1001	110000 0110
K28.2	001111 0101	110000 1010
K28.3	001111 0011	110000 1100
K28.4	001111 0010	110000 1101
K28.5	001111 1010	110000 0101
K28.6	001111 0110	110000 1001
K28.7	001111 1000	110000 0111
K23.7	111010 1000	000101 0111
K27.7	110110 1000	001001 0111
K29.7	101110 1000	010001 0111
K30.7	011110 1000	100001 0111

Ordered Sets are combinations of Special Characters and Data Characters used for link initialization, and frame delineation. Ordered Sets can be one, two or four characters long. The Ordered sets used by Cassini+ is specific to 802.3 and is currently defined as:

Table 7-2 802.3z Ordered Sets

Code	Function	Ordered Set
C1	Link Configuration 1	K28.5 D21.5 **configuration**
C2	Link Configuration 2	K28.5 D2.2 **configuration**
I1	Idle/Flip Disparity	K28.5 D5.6
I2	Idle/no flip	K28.5 D16.2
S	SOP	K27.7
T	EOP1	K29.7
R	EOP2	K23.7
H	EOP invalid	K30.7

Where **configuration** carries a 16 bit value exchanged during the Auto-negotiation phase (normally identical to the PCS MII Advertisement Register).

7.2.3 PCS link initialization and Auto-negotiation

The PCS function is also responsible for link initialization, including Auto-negotiation. The initialization and Auto-negotiation functions apply to SERDES and Serialink interface modes. Whenever the interface mode is MII, PCS Auto-negotiation is also attempted by the PCS over Serialink. This enables the software driver to implement a *link status* based switch-over for Cassini+ applications with MII and Serialink interfaces. The interface that has a cable connected and link UP is automatically selected.

7.3 Diagnostic LEDs

LEDs intended to serve as installation aid comprise:

- LINK OK
- FULL DUPLEX
- TRANSMIT ACTIVITY
- RECEIVE ACTIVITY
- COLLISION

Link OK and Full Duplex conditions are reported by the PCS and only apply to configurations that use Cassini+'s 8B/10B PCS function. For MII/GMII modes the software driver may activate the Link and Full Duplex LEDs using the XIF Register. It is also possible to rely on the MII transceiver LEDs for these modes.

The remaining LEDs are logically derived from the MII control signals and apply to MII/GMII, SERDES and Serialink modes.

The Transmit LED is derived from TXEN active. The Receive LED from RXDV, and the Collision LEDs from the COL signal.

All LED signals except LINK OK and FULL DUPLEX are pulse stretched to at least 100msec duration.

This Page Intentionally Left Blank

Interfaces and Data Paths 8

8.1 Interfaces and Data Paths

8.1.1 PCI Interface

Basic Feature Set

The major features of the PCI interface are:

- PCI interface compliant with PCI spec revision 2.2
- 64 bit data path master interface
- 66.6 MHz bus clock operation. Bus interface will run at lower frequencies up to 66.6 MHz (e.g. 33.3 MHz) with lower bus interface performance.
- Supports 32 bit bus when installed in a 32 bit bus slot
- 32 bit data path slave interface.

Universal IO

Operates in 5V and 3.3V slots.

8.1.2 Bus Interface Module Signal Definition

The Gigabit-to-PCI Interface (GPX) protocol bus signals are grouped into two sets needed for D(V)MA read or write operations and are defined below.

8.1.2.1 GPX Tx Interface Signals

The set of signals used for D(V)MA operations pertaining to the transactions between Tx and the PCI bus follow:

- txdma_rd_req (Core output)

  Core asserts txdma_req when it desires to initiate a read transaction. It keeps txdma_rd_req high until BIM asserts txdma_rd_ack.

- txdma_rd_ack (Core input)

  BIM asserts txdma_rd_ack (one clock width only) when it has captured all pertinent cycle information (address, size, type). Core can remove that information and present next cycle information. BIM is capable of queueing upto 3 read requests . Further requests, BIM limits the number of read transactions pending by delaying assertion of txdma_rd_ack.

- txdma_rd_addr[63:0] (Core output)

  This is the dma read address lines. TX places the address first on this bus. After getting the txdma_rd_ack from BIM, TX can place change the address for the next read request. The address is byte resolution. However, data transfer across the GPX bus is always 64 bit aligned.

- dma_rd_data[63:0] (Core input)

  This is the dma read data. As mentioned above, the data returned is aligned to a 64 bit address in little endian. If all descriptors and data buffers are aligned by the driver on that basis to begin with, there is no issue here. If the alignment of data structures in host memory is arbitrary the core logic must extract the data from the 64 bit wide bus. This bus is shared with Rx.

- txdma_rd_size[13:0] (Core output)

  This indicates the number of bytes the TX wishes read transfer in the current transaction. Rather than have an explicit infinite burst encoding, the exact data transfer size in bytes is given to BIM and it optimally decides how to handle the transaction taking into account infinite burst, cache line size, best PCI memory read operation to use, etc.

- txdma_rd_type[1:0] (Core output)

  This signal determines the type of read. Descriptor read and buffer read are the two types of reads implemented. This signal is sampled by BIM along with the address and size information.

- txdma_rd_rdy (Core input)

  This input indicates to the TX that the requested read data is avilable on the txdma_rd_data line. During read transfer, this signal means that TX can sample the data when txdma_rd_rdy is asserted on a rising edge of the core clock.

- txdma_wr_req (Core output)

  Core asserts txdma_req when it desires to initiate a write transaction. It keeps txdma_wr_req high until BIM asserts txdma_wr_ack.

- txdma_wr_ack (Core input)

  BIM asserts txdma_wr_ack (one clock width only) when it has captured all pertinent cycle information (address, size, type). Core can remove that information and present next cycle information. BIM limits the number of read transactions pending by delaying assertion of txdma_wr_ack.

- txdma_wr_ad[63:0] (Core output)

This is a multiplexed line for dma write address and data. When the TX is making a request for write, the BIM interprets this as address. However, when the BIM is ready to transfer data from TX , these lines contain data. TX places the address first on this bus. The address is byte resolution. However, data transfer across the GPX bus is always 64 bit aligned.

- txdma_wr_size[13:0] (Core output)

  This indicates the number of bytes the TX wishes write transfer in the current transaction. Rather than have an explicit infinite burst encoding, the exact data transfer size in bytes is given to BIM and it optimally decides how to handle the transaction taking into account infinite burst, cache line size, best PCI memory write operation to use, etc.

- txdma_wr_type[1:0] (Core output)

  This signal determines the type of write. Completion writeback is the only write operation initiated by TX.. This signal is sampled by BIM along with the address and size information.

- txdma_wr_rdy (Core input)

  This input indicates to the TX that BIM is ready to get the write data on the txdma_wr_ad lines.

- txdma_wr_xfr_done (Core input)

  BIM asserts this signal for one clock indicating it has transferred all data affiliated with the Tx completion writeback request. This signal indicates that the write transfer is completed successfully over the PCI to the host memory.

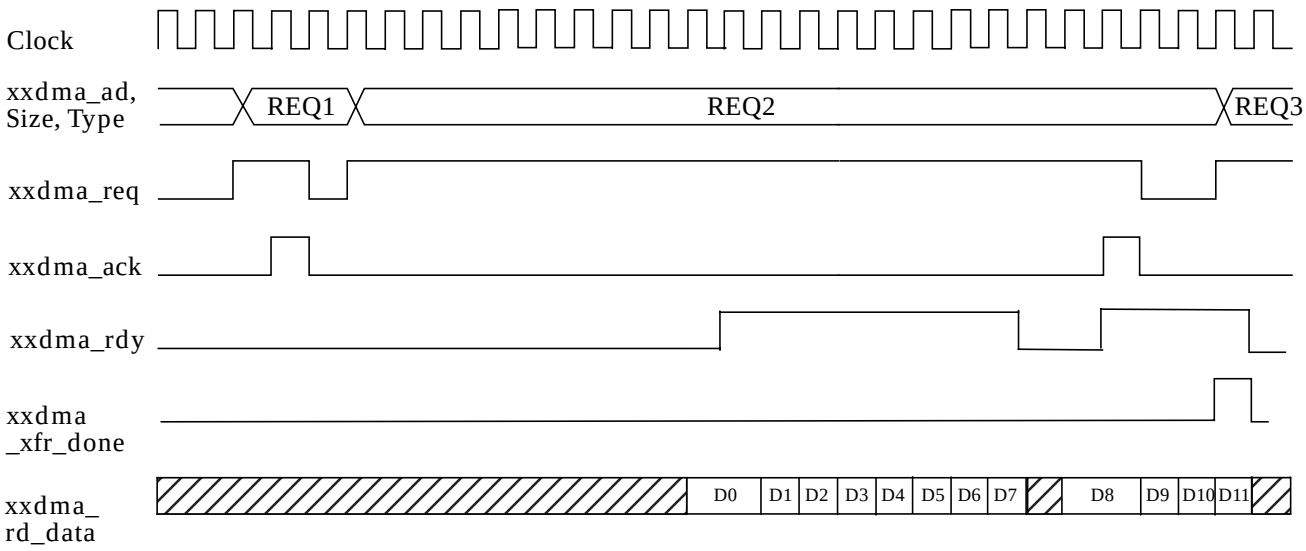
8.1.2.2 *Type Field Encoding*

The type and fields use a two bit encoding specifying the type of access being made by the TX or RX units.

- 00 : data buffer access
- 01 : free buffer descriptor access
- 11 : completion ring access

8.1.2.3 *Read Cycle Example*

An example read cycle is shown below: (same for Tx and Rx)



Note that BIM contains N (N=3 in Cassini+) 64 byte buffers in the read buffer synchronization ring. For a read operation which exceeds 192 bytes and prior to which the buffers are empty, BIM can burst read in 192 bytes from host memory without need of deasserting IRDY# or terminating the bus cycle to resume it later. The core reads 64 bytes once it receives a dma_rdy assertion from BIM. After reading the last 8 bytes from the buffer or simultaneous with that read, the BIM continues with the next buffer if it is full. Otherwise, it introduces a wait state by deasserting the dma_rdy. When the dma_rdy is asserted again, the first buffer becomes free for BIM to deposit more read data. This allows two things. First, a continuous read across 64 byte buffer boundaries with no core domain null cycles is provided in this protocol when BIM is reading PCI data in at a rate comparable to the core's rate of data reception or when BIM starts to read data prior to the core being ready initially.

8.1.2.4 GPX Rx Transaction Signals

The set of signals used for D(V)MA write operations over the PCI bus follow:

- rxdma_req (Core output)

  Core asserts rxdma_req when it has a write operation desired. It keeps rxdma_req asserted until rxdma_ack is asserted by BIM.

- rxdma_ack (Core input)

  BIM asserts rxdma_ack (one clock) when address, size and type have been latched allowing the core to prepare for the next transaction.

- rxdma_ad[63:0] (Core output)

Multiplexed address and data for the DMA transfer. The data is for the DMA write direction. Address is removed and changed to data by HRP, after getting the rxdma_ack from BIM. Address is to byte resolution. The bus interface starts the write operation at the address indicated by HRP. The data has to be alingned within a 64 bit field in consistent with the starting address of the burst in little endian format.

- dma_rd_data[63:0] (Core output)

  Data in the read direction for the core. This bus goes to Rx and Tx.

- rxdma_size[13:0] (Core output)

  Bytes to be written at minimum unless a cache line necessitates junk writes by the bus interface.

- rxdma_type[1:0] (Core output)

  This signal determines the type of transaction. All three type definitions will be used by Rx. This signal is sampled by BIM along with the address and size information.

- rxdma_rd (Core output)

  This line from the Rx block indicates whether a read or the write transaction is requested.A HIGH indicates a read transaction and a LOW indicates a write transaction.

- rxdma_rdy (Core input)

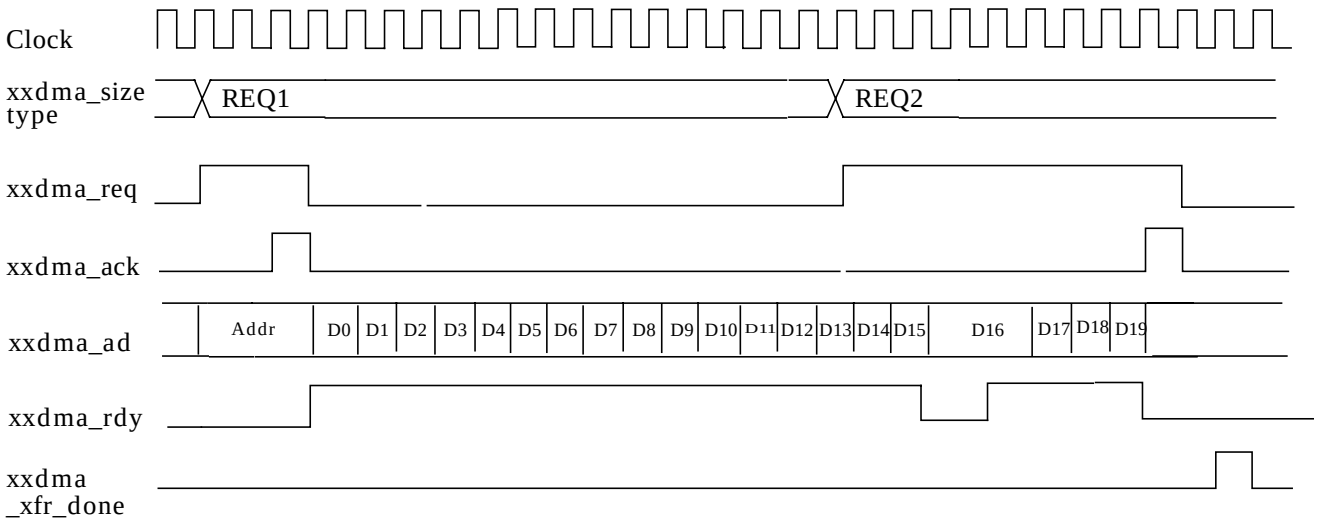
  This signal indicates that BIM is ready to transfer data across the GPX bus. During read transfer, this signal means that Rx can take the data when rxdma_rdy is asserted on a rising edge of the core clock. On the other hand, during a write transaction, this indicates that BIM can take the data at the rising edge of core clock from the Rx.

- rxdma_xfr_done (Core input)

  After the PCI bus logic completes the last write for this completion descriptor request on the PCI bus interface, it asserts this signal informing the core that the data has completely been transferred over the PCI bus. This allows the core to synchronize interrupts or other processes with the I/O bus state.

8.1.2.5 *Write Cycle Example*

An example write transaction is shown below. (same for Rx and Tx)



8.1.3 MII/GMII Interface

As an MII this interface can be used with 100BASE-T transceivers. The implementation of the interface conforms with the MII specification. The MII data path is defined as two uni-directional nibble-wide buses: one for transmit and one for receive, with transmit and receive clocks supplied by the transceiver. The MII is clocked at 25MHz or less, and may be connectorized.

GMII is an extension of the MII that retains the logical functionality of its control signals, while it widens the data path to two uni-directional byte-wide buses, and it adds support for *carrier extension* characters. In addition to the transceiver supplied clocks, the MAC provides a transmit clock output that may be used by the transceiver to sample transmit data and control signals with less stringent clock skew requirements. The GMII is clocked at 125 MHz or less, and is an inter chip interface with no intervening connectors.

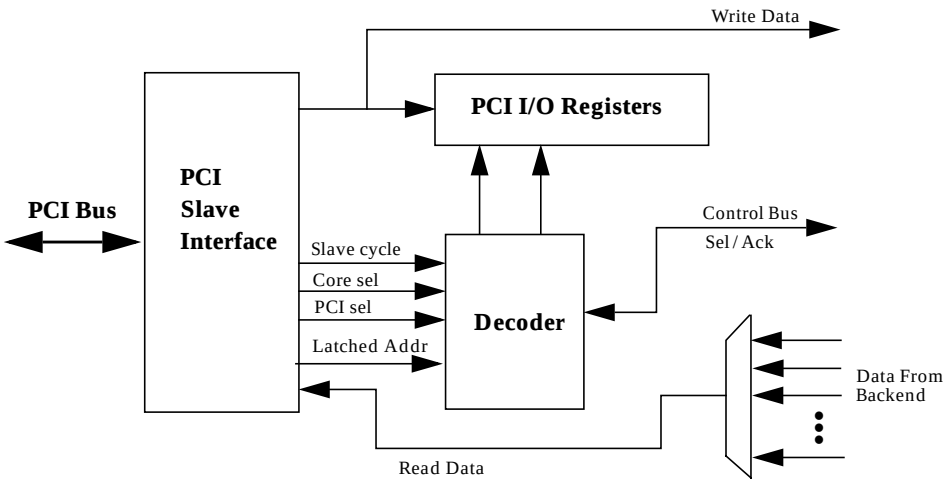
The MII/GMII managment interface utilizes a 1-bit wide bi-directional serial line.

8.1.4 Expansion ROM Interface

A byte-wide PROM may be used as a Cassini+ 's expansion ROM. This device would typically store the Fcode image as well as the unique MAC address for the board. The maximum PROM size is 64kbytes. The interface only supports PROM reads. The MAC address is written into the PROM at ICT time as part of the PCI expansion ROM VPD structure.

8.1.5 PIO Bus Interface

The PIO accesses are addressed by the PCI slave interface and gets converted into the core clock domain for either writing into the registers or to get data from the registers. The data flow during the PIO accesses from the PCI bus to the backend and vice versa is depicted in the following diagram.



During the PIO cycles the PCI slave functions as a posted write or a read ahead register. The PCI cycle is signalled a retry to the master initiating the cycle until the write or read operation is completed in the GPX bus. The PCI slave cycle is converted into a sub-slave cycle on the GPX bus. Since the data has to be transferred across two different clock domains, this backend bus operates using the “select” and the “ack” modes. The PCI cycle is decoded to be addressed to one of the various blocks in the core logic. (e.g. Rx, Tx, MAC, etc.) The write data is places in a common bus. Since the select and ack are separate for each of the blocks, the respective block samples the data using its own select signal. During the read cycle each of the blocks return the read data on a separate bus. The appropriate data is selected by the PCI slave logic.

For the PCI related resources the PCI slave blocks respond with the relevant data. For the register accesses in the core side of Cassini+ , the data flows across the two different clock domains. The request for the PCI slave logic is passed to the respective block in the core side as a request for read or write. Along with request and the command, the address is also passed. In the case of a write cycle the 32 bit data is also sent to the backend. There are individual request and ack lines for each of the blocks in the core side. The write data is common to all the blocks and the respective block has to sample the data on seeing the request. BIM will use the ack from the respective block as the cycle-end signal. In the case of write cycle, BIM will remove data after getting the ack from core blocks. Similarly in the case of read cycles, BIM will sample the data when ack is valid. If the ack signal from the core is delayed due to any reason during the synchronization, the PCI slave will retry the cycle until a valid data exchange happens.

The following section described the signals on the core side for the slave interface.

- slv_addr[19:0]

These are the lower bits of the PCI address for the slave access. The backend blocks can sample the range of address corresponding to their internal registers and have a decode logic. The address is valid when the request signal is active.

- slv_wr_data[31:0]

The data from the PCI bus corresponding to the slave cycle. This data goes to all the blocks in the core and the blocks have to sample the data to write into the registers.

- tx_slv_data[31:0], rx_slv_data[31:0], mac_slv_data[31:0], pcs_slv_data[31:0]

This is the data to be returned to the PCI bus in response to the read cycle. Each block in the core has separate data lines. Depending on the target and using the ack signal, BIM will take the data corresponding to the selected block. After activating the slv_ack signal, the respective blocks can remove the data from this bus.

- slv_rd

The read / write indication for the slave access for the core blocks. When HIGH, this indicates a read and a LOW indicates a write cycle. This signal is valid along with the request.

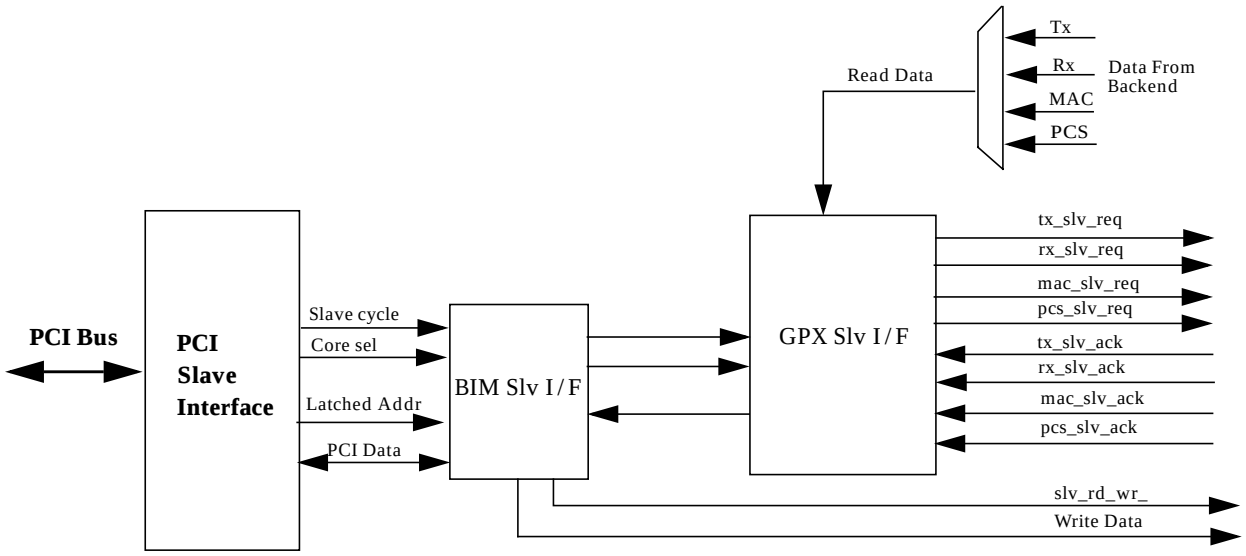
- tx_slv_req, rx_slv_req, mac_slv_req, pcs_slv_req

PIO request for the HTP, HRP, MAC, and PCS blocks respectively. This is an active high signal and it stays active until the ack is received.

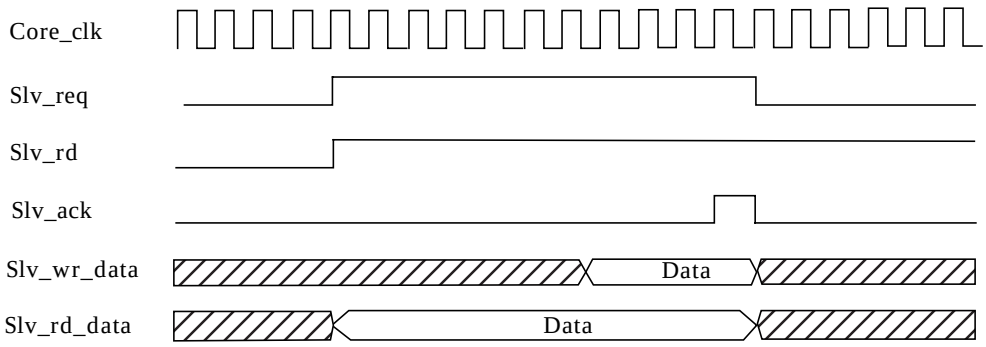
- tx_slv_ack, rx_slv_req, mac_slv_ack, pcs_slv_ack

These are the ack from the core blocks in response to the PIO accesses. This signal of one core clock duration so that when this signal is valid, the data is qualified for the read or write transaction.

The following diagram gives addition insight to the slave path. The signal names on the core side are also show in this diagram.



An example slave transaction is shown below



8.1.6 RxDMA <-> MAC Interface

This non-exposed interface is utilized for transferring incoming packet data from the synchronization fifo in the RxMAC to the receive fifo in the RxDMA. All signals are active high.

Signal Description

- **rx_data[63:0]**
  This data bus is used to transfer eight bytes of receive packet data or packet status from the RxMAC to the RxDMA. These lines shall present the data to the RxDMA in a “little endian” format.

- **rx_tag**
  This signal is used to indicate the completion of a packet transfer from the RxMAC to the RxDMA. For each receive packet, the last data double-word and the status double-word shall have this bit set. The remaining double-words of the packet shall have this bit cleared.
- **rx_mac_req**
  This signal is driven by the MAC and is used to indicate to the RxDMA that the MAC is ready to start the transfer of the next burst of data. The rx_mac_req should deassert the cycle after the MAC receives the rx_ack. Otherwise the MAC is assumed to be signalling a new request by extending a previous request. Once exception to this is when the MAC is transferring the status as the only word. It may keep the rx_mac_req asserted for one cycle past the deassertion of the rx_ack signal. This cycle of rx_mac_req should be ignored by Rx. Unlike other cases, it will not be treated as a new request.
- **rx_ack**
  This signal is driven by the RxDMA and is used to indicated to the MAC that the RxDMA is executing a burst transfer.
- **tx_fc_xoff_req**
  This signal is driven by the RxDMA and is used to indicate to the MAC that a “Pause” Flow Control frame should be transmitted on the medium, with a pause parameter as programmed in the Pause Value Register in the CMAC.
- **tx_fc_xon_req**
  This signal is driven by the RxDMA and is used to indicate to the MAC that a “Pause” Flow Control frame should be transmitted on the medium, with a pause parameter equal to zero.
- **tx_fc_ack**
  This signal is driven by the MAC, and is used to indicated to the RxDMA that the MAC has transmitted the “Pause” Flow Control frame previously requested by the RxDMA using the tx_fc_xoff_req/tx_fc_xon_req request lines.
- **pkt ready**
  This signal is used for the packet bursting feature and is valid only in the Half Duplex mode. It indicates whether there is at least 2 packets available in the Tx fifo. Whether it is asserted or not MAC sends ’tx_mac_req’ signal to txdma before accepting any new packets

Protocol Description

Data from a receive packet that arrives from the medium is first stored in the synchronization fifo in the RxMAC. After enough data has been accumulated to execute a burst (32 bytes) or the end of the packet has been detected, the MAC asserts the request line to the RxDMA (rx_mac_req).

Upon detection of a request, and if the receive queue can accept the burst, the RxDMA shall assert the acknowledgment (rx_ack) **exactly one clock** after the assertion of the request. Once set, it shall remain asserted **for exactly 4 clocks**, if a full 32-byte burst transfer is executed. If the end of the packet is detected in the middle of a burst, the RxDMA **shall de-assert** the rx_ack after the transfer of the status word has been completed. In response to rx_ack, the MAC **may or may not** de-assert the request line.

Upon detection of rx_ack, the MAC will start driving the first eight bytes of valid data on the rx_data lines at the next rising edge of the local clock, and will switch to the next two consecutive bytes of the packet for three more clock edges. Thus, the acknowledgment signal performs framing of a 32-byte burst transfer between the MAC and the RxDMA.

At the end of a packet transfer, the MAC appends to it a status/control word after the last word that contains at least one valid byte of data. The “invalid” bytes in the last data word of the packet are driven to 0x00 by the MAC.

Upon detection of congestion in the receive data path, the RxDMA may assert the tx_fc_xoff_req signal to the MAC, requesting a transmission of a flow control pause frame to the remote end of the link. The pause time is expected to be programmed by the software in the Pause Value Register. The MAC will complete the transmission of the frame that is currently in progress, transmit the pause frame (when the transmit link becomes free) and acknowledge the request. In response to tx_fc_ack, the RxDMA shall de-assert the tx_fc_xoff_req signal.

Once the RxDMA detects that the receive data path is no longer congested, it may assert the tx_fc_xon_req signal to the MAC, requesting the transmission of a flow control pause frame with the pause time equal to zero. This effectively enables the transmission from the remote end of the link. The MAC will complete the transmission of the frame that is currently in progress, transmit the pause frame and acknowledge the request. In response to tx_fc_ack, the RxDMA shall de-assert the tx_fc_xon_req signal.

Status Word Format

The format of the receive packet status word, generated by the MAC, is as follows:

[63]: Abort: Indicates to the RxDMA that the packet that is currently being transferred is not valid.

[62]: Bad_Packet: When set, this bit indicates that the receive packet is a bad packet based on either: a CRC mismatch detected by the RxMAC, or invalid coding reported by the PCS block.

[61]: Reserved: Set to 0.

[60]: Hash_Pass: When set, this bit indicates that the received packet’s DA has passed the hashing filter algorithm in the RxMAC.

[59:44]: Hash_Value: This field provides the value of the received packet’s hashed DA, as computed by the RxMAC.

[43]: Reserved: Set to 0.

[33:30]: Alternate Address Filter Pass:

Indicates the result of the Destination Address matching of the unique or alternate addresses (stored in MAC Address 1 - 47 Registers).

[29:16]: Pkt_Length: This field provides the length of the received packet (in bytes), as computed by the RxMAC.

[15:0]: Reserved: Set to 0.

8.1.7 IPP <-> Header Parser Interface

This non-exposed interface is utilized for transferring incoming packet data from the input port process fifo in the Rx to the fifo in the Header Parser and transferring control information from the Header Parser to the input port process. All signals are active high.

Signal Description

- **hp_status_rdy**
  This signal is driven by the Header Parser and is used to indicate to the IPP that contents on the hp_status_data bus is ready.
- **hp_status_ack**
  This signal is driven by the IPP and is used to indicate to the Header Parser that the IPP has used the hp_status_data and doesn't require it any longer.
- **hp_status_data[8:0]**
  This data bus is used to transfer checksum start offset, Layer 3 header offset, SAP and load balancing CPU information for the packet from the Header Parser to the IPP.
- **hp_flow_code_rdy**
  This signal is driven by the Header Parser and is used to indicate to the IPP that contents on the hp_flow_data bus is ready.
- **hp_flow_code_ack**
  This signal is driven by the IPP and is used to indicate to the Header Parser that the IPP has used the hp_flow_data and doesn't require it any longer.
- **hp_flow_data[74:0]**
  This data bus is used to transfer flow id and opcode information for the packet from the Header Parser to the IPP for input into the Rx Control Fifo.

Protocol Usage Description

Data from a receive packet that arrives from the medium is first stored in the input port process fifo in the Rx. After an entire packet of data has been accumulated, the IPP waits for the Header Parser to provide the status word over the hp_status_data bus. The IPP will then begin transfering data to the Rx Fifo. Afterwards the IPP will wait for the flow_code information from the Header Parser on the hp_flow_data bus. When this is ready it will merge this data with checksum data to create an entry in the Rx Control Fifo.

Header Parser Flow Data Format

The format of the hp_flow_data bus, generated by the Header Parser, is as follows. This will be added as an entry to the Rx Control Fifo.:

[74:61] TCP Segment size :Total size of the TCP packet (header plus payload). The IPP uses it to do checksum on reassembly packets.

[60:45]: SAP : Captured from the Ethernet type field. The driver will use this information to sort packet types.

[44:38]: Layer 3 Header Offset :

Indicates where the Layer 3 header in the packet begins. The driver will use this information to find header boundaries.

[37:32]: Load Balancing CPU Number:

Indicates which CPU the packet should be sent to be processed. This is a function of the number of CPU's available per machine and a hash on the packet flow key.

[31]: No Hardware Assist:

Indicates that the IPP parser was unable to successfully parse this packet. The packet will enter a slow path and incur no hardware reassembly assistance.

[30] Force Flow Lookahead:

Indicates that the HRP should force the release flow bit to equal the flow_lookahead value instead of its normal value for opcode 2. This is to support batching without reassembly.

[29:27]: Opcode: The Flow Database Manager within the IPP categorizes the packet into one of eight criteria. This opcode indicates how the packet should be processed by the HRP. This information will be incorporated by the IPP into the control packet and entered into the Rx Conrol Fifo.

[26:21]: Flow ID : A six bit ID associated with each parsed packet. This information will be used by the HRP to index the DMA Reassembly Table.

[20:14]: TCP Payload Offset:

A pointer to the beginning byte of the TCP payload in the packet.

[13:0]: TCP Payload Size :

Indicates the length of the TCP Payload. The HRP uses this to do reassembly.

Header Parser Status Data Format

The format of the hp_status_data bus, generated by the Header Parser, is as follows:

[8]: Force drop:

Tells the IPP to drop the packet. This is used for magic packet filtering. All non-essential packets as determined by the microcode will be marked for drop.

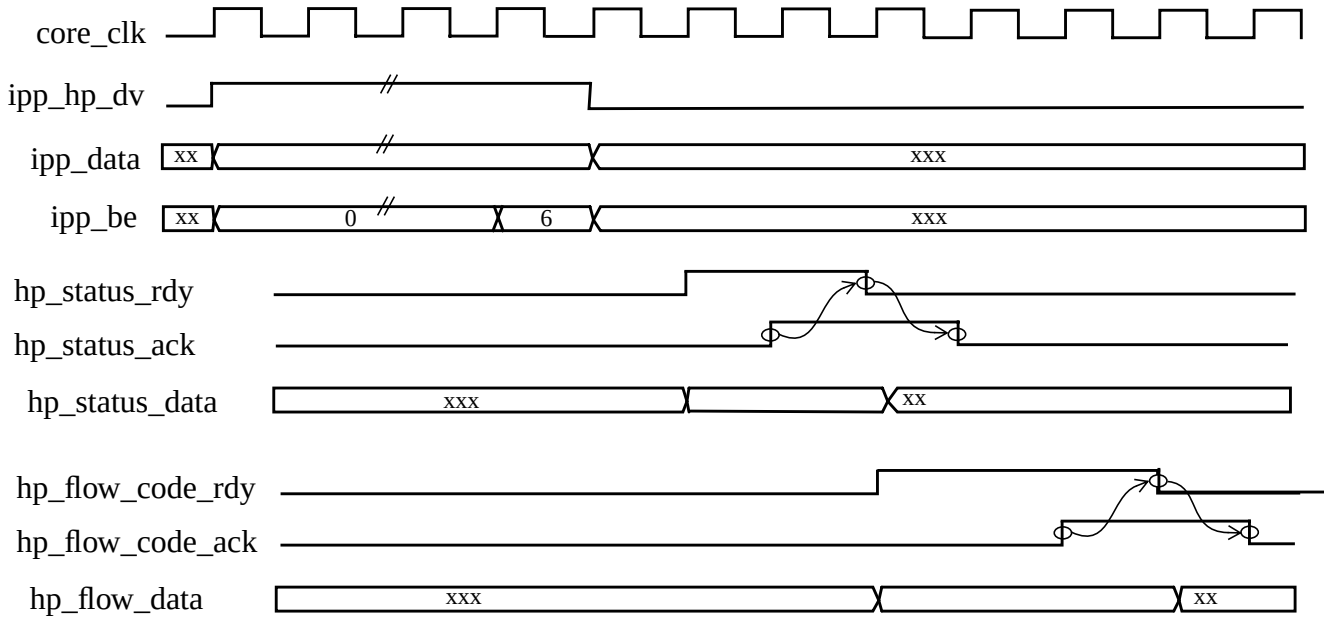
[7]: Control packet:

Tells the IPP whether the packet is a control packet or not. This is used by the Random Early Detection circuitry to determine whether the packet is a candidate for drop or not. A control packet is one which has a TCP data size of zero or has either the SYN or FIN flags set.

[6:0]: Checksum Start Offset :

Tells the IPP Checksum generator at which byte to begin the checksum calculation. It is given in a 2 byte granularity. Checksum is always done to the end of the packet.

Figure 8-1 Timing Diagram of IPP <-> Header Parser Interface



8.1.8 TxDMA <-> MAC Interface

This non-exposed interface is utilized for transferring the outgoing packet data from the TX FIFO in the TxDMA to the synchronization fifo in the TxMAC. All signals are active high.

Signal Description

- **tx_data[63:0]**
  This data bus is used to transfer 8 bytes of transmit packet data or packet status from the TxDMA to the MAC. These lines shall present the data to the MAC in a “little endian” format.
- **tx_tag**
  This signal is used to indicate the completion of a packet transfer from the TxDMA to the MAC. For each transmit packet, the last data double-word and the status double-word shall have this bit set. The remaining double-words of the packet shall have this bit cleared.
- **tx_mac_req**
  This signal is driven by the MAC and is used to indicate to the TxDMA that the MAC is ready to start the next burst of data transfer.
- **tx_ack**
  This signal is driven by the TxDMA and is used to indicated to the MAC that the TxDMA is executing a 32-byte burst transfer.
- **tx_mac_retry_req**
  This signal is driven by the MAC and is used to indicate to the TxDMA that the MAC is requesting a re-transmission of the current packet from the beginning, due to a collision on the shared medium.
- **tx_pkt_rdy** This signal is driven by the TxDMA and is used to indicate to the MAC that a burst should be occuring.

Protocol Description

After the MAC completes the transmission of a packet on the medium, it asserts the tx_mac_req line to the TxDMA. If at least one packet is ready to be transmitted, the TxDMA asserts the tx_ack **exactly one clock** after the assertion of the request. Once set, it shall remain asserted **for exactly 4 clocks**, if a full 32-byte burst transfer is executed. If the end of the packet is detected in the middle of a burst, the TxDMA **shall de-assert** the tx_ack after the transfer of the control/status word has been completed. In response to tx_ack, the MAC **may or may not** de-assert the request line.

Simultaneously with the assertion of the acknowledgment signal, the TxDMA starts driving the first eight bytes of valid data on the tx_data lines. It will switch to the next eight consecutive bytes of the packet at the next rising edge of the local clock, and will repeat this action for two more consecutive clock edges. Thus, the acknowledgment signal performs “framing” of a 32-byte burst transfer between the TxDMA and the MAC.

At the end of a packet transfer, the TxDMA appends to it a status/control entry after the last double-word that contains at least one valid byte of data.

If during a packet transfer the TxMAC encounters a collision on the medium, it asserts the tx_retry_req signal together with the tx_mac_req. In response, the TxDMA prepares itself for packet re-transmission by rewinding its Read Pointer to the beginning of the current packet plus 32 bytes, and asserts the tx_ack signal **within a maximum of 4 clock cycles** after the assertion of tx_mac_req.

The MAC may assert the tx_retry_req signal at any time during the transfer of the data portion of the packet, or after the entire packet has been transferred to the MAC. The TxDMA must monitor the tx_retry_req signal of the first burst of a packet transfer to the MAC in order to determine whether the previous packet has been successfully transmitted or needs to be re-transmitted.

Control Word Format

The format of the transmit packet control word, generated by the OPP, is as follows:

[63]:	Abort:	Indicates to the TxMAC that the packet that is currently being transferred for transmission is invalid. The Tx_MAC will generate a TX_ER indication on the MII. This bit is always set to zero in Cassini+ .
[62]:	No_CRC:	Indicates to the TxMAC not to generate a CRC field for the current packet. In Cassini+ this bit corresponds to the “No CRC” bit in the TX Descriptor.
[61:59]:	LBB:	Last Byte Boundary. This field indicates to the TxMAC the position of the last valid byte of data in the previously transferred last data word.
[58: 0]:	Reserved:	This field is ignored by the TxMAC.

This Page Intentionally Left Blank

Interrupts and Error Handling 9

9.1 Interrupt Logic Structure

Cassini+ uses a single interrupt line to signal events that require host intervention. The specific interrupt events are defined by a two-level interrupt status structure. The main events have their corresponding bits in the interrupt status register, while secondary events are reported by secondary status registers whose bits share an entry in the interrupt status register.

Enabling (masking) and clearing of every interrupt condition is done at its own status register level. No additional restrictions exist for polling the interrupt bits at any time. The main interrupt status register may be read through two locations, one of them automatically clears all its active interrupts upon reading it, while the other location allows reading with no side effects. Secondary status registers are cleared upon reading them.

The receive interrupt is the main condition used by the software driver to remove frames from the host receive queue and pass them to the upper layers. If the receive frame arrival rate lags behind the frame processing rate, the receive interrupt will be activated for every packet, subject to a blanking mechanism. If the receive frame arrival rate is faster, the interrupt handler may process more than one frame per interrupt. The receive interrupt blanking mechanism may be used to delay the receive interrupt assertion until a programmable number of receive packets were received, or a minimum time has elapsed since the last receive interrupt.

Transferring frames between the upper layer and Cassini+ is not interrupt driven, however transmit interrupts may be enabled to assist in the scheduling and buffer reclaim of the transmit process. Transmit interrupts may be requested on any arbitrary frame(s) using the TX_INT_ME event, and will be generated when the specific frame(s) is completely transferred from the host queue to the TX FIFO.

The interrupt structure is defined with the guidelines of:

1. Minimizing dependencies between receive and transmit activities
2. Minimize number of slave reads required per interrupt

Several other interrupts conditions exist. Some of them are associated with error conditions, maintaining statistics, or reflecting a change in the state of the link or transceiver. The Receive Buffer Overflow interrupt indicates the reception of a frame that was longer than the receive buffer allocated by the driver.

The interrupt status structure is illustrated in Figure 9-1. The detailed interrupt conditions and register definitions are specified in the Programmer's Model Chapter.

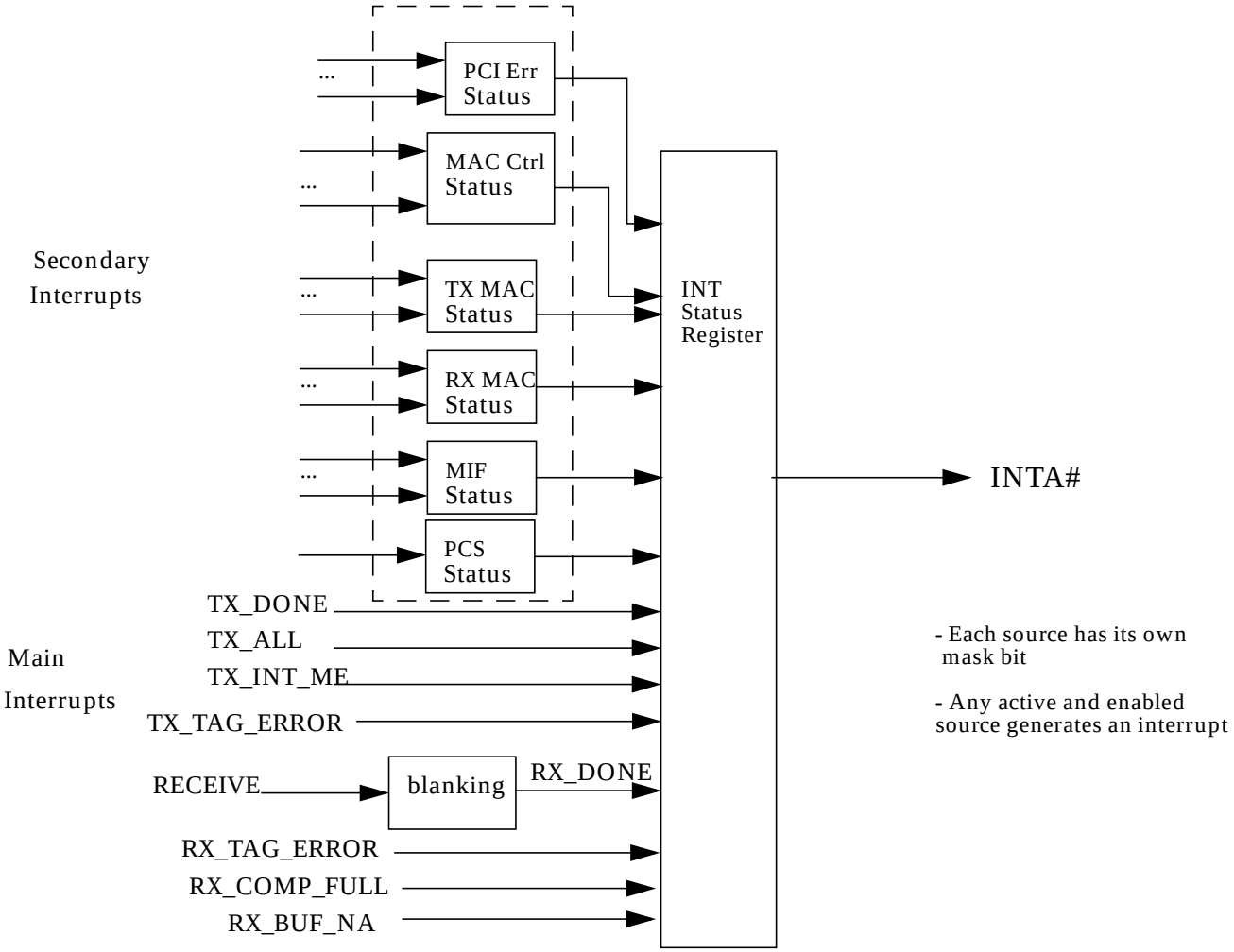


Figure 9-1 Interrupt Structure

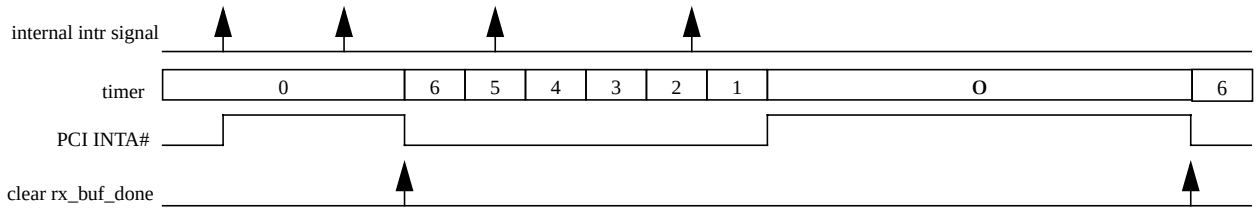
9.1.1 Receive Interrupt Blanking

The RX_DONE receive interrupt can only be set if at least one of the following conditions is true:

1. The number of packets received since the last time the interrupt status register was read is at least RX_INTR_PACKETS.
2. The time elapsed since the last time the interrupt status register was read is at least RX_INTR_TIME. The clr_rx_buf_done signal comes from the BIM to indicate when the rx_buf_done interrupt is cleared. This can occur either during the read of the interrupt status register or a write of the respective bit in the selective clear interrupt register.

The values of RX_INTR_PACKETS and RX_INTR_TIME are programmed in the RX Blanking Register. RX_INTR_TIME is specified in units of System clocks divided by 512.

Figure 9-2 **Cassini+ interrupt blanking count down example**



9.2 Error Conditions and Recovery

There are two types of error conditions that can be encountered during the normal operation: fatal errors and non-fatal errors.

Fatal errors are errors that should never occur. They usually indicate a serious failure of the hardware or a serious programming error. When this type of error occurs, the recovery process is non-graceful and involves software reset after the appropriate actions were taken to correct the failure. Fatal error events are always reported to the software via interrupts.

Non-fatal errors are errors are that are expected to occur when certain conditions occur on the network or in the system. When this type of error occurs, a graceful recovery mechanism is provided via a combination of hardware and software as described below. Non-fatal errors may or may not be reported to the software.

9.2.1 Non-Fatal Errors

Rx buffer not available

This condition is encountered when no more receive descriptors are available for receive frames. An interrupt is generated to the device driver to indicate the occurrence of this event. The RX DMA clears this condition as a result of the host CPU updating the RX Kick Register after posting new buffers.

Due to flow control, and the RX FIFO size, this condition does not necessarily mean that receive data is being discarded. Data discarding is reported by the *Rx FIFO overflow* condition.

Rx completion ring full

This condition is encountered when no more completion entries on the completion descriptor ring are available for receive frames. An interrupt is generated to the device driver to indicate the occurrence of this event. The RX DMA clears this condition as a result of the host CPU updating the RX Completion Tail Register after processing more completion entries.

Due to flow control, and the RX FIFO size, this condition does not necessarily mean that receive data is being discarded. Data discarding is reported by the *Rx FIFO overflow* condition.

Rx buffer overflow

The RX DMA transfers frames from the buffer memory to the host memory. Cassini+ will not detect if the size of a buffer available in the host memory is smaller than the frame size, therefore software is responsible for allocating buffers large enough to fit the value programmed in the *Maximum Frame Size Register* in the MAC. Frames that exceed this value are truncated by the MAC, and the buffer re-claimed using the Rx Abort mechanism described below. Software can determine if such longer frames exist on the network and are directed to Cassini+ by reading the *Length Error Counter.*

Rx Abort from the MAC

A receive frame may be aborted while in transit from the network to host memory. Depending on how far into the frame the abort condition is detected, different blocks might be involved in the recovery.

In some cases the condition is detected early on by the RX MAC, even before the frame was presented to the RX FIFO interface. In this case the frame is silently dropped by the RX MAC with no RX DMA involvement.

If the abort condition is detected after transfer to the RX FIFO has begun, it is signalled by setting the “abort” bit in the status word. If the RX DMA observes the “abort” bit set before it has started removing the frame from the RX FIFO, the frame will be discarded by rewinding the RX FIFO Write Pointer back to the value of the Write Shadow Pointer. Otherwise, when it is too late to rewind the Write Pointer and the RX DMA is responsible for discarding the frame and re-using its descriptor.

RX Abort conditions are not reported as such to the host, but the events causing it (BAD CRC, fragment, RX_ER, RX FIFO Overflow, etc.) have their own reporting mechanisms.

Bad frame reception, for troubleshooting and network monitoring purposes, may be enabled by setting the Err_Check_Disable bit in the *RX_MAC Configuration Register*. In this case the “abort” bit in the status word is never set. All bad frames are received, including fragments shorter than the value programmed into the *Minimum Frame Size Register*.

Rx FIFO overflow

If the RX FIFO becomes unable to receive more data from the RX MAC, in spite of the Flow Control mechanism, this condition propagates to the RX MAC. The RX MAC will discard data when it runs out of space in its synchronization FIFO, and optionally generate an interrupt by setting the Rx_Overflow bit in the *RxMAC Status Register*. Partial frames resulting from this condition are discarded by setting the “abort” bit in the status word.

9.2.2 PCI Error handling

Cassini+ ’s PCI Error handling follows section 3.8 of the PCI Spec revision 2.1, and allows recovery at the device driver level, whenever possible, by using the PCI Error Status Register when fatal errors occur.

PCI Bus Errors, like parity errors, may result in fatal or non fatal errors, depending on the nature of the error and the programming of PCI Configuration space bits. The bits involved are:

Parity Error Response - PCI Command Register, bit 6

SERR# Enable - PCI Command Register, bit 8

Detected Parity Error - PCI Status Register, bit 15

Signaled System Error - PCI Status Register bit, 14

Data Parity Error Detected - PCI Status Register, bit 8

Even parity generation is the responsibility of the agent that drives the PCI address/data bus for any given phase. Cassini+ 's PCI Bus parity error response is determined by the Parity Error Response and SERR# Enable bit values programmed in PCI Configuration Space.

9.2.2.1 PCI Slave Bus Parity Errors

Slave parity errors are non-fatal and are reported using the Set Detected Parity Error bit in PCI Configuration space. Table 9-1 defines Cassini+ 's response to slave access parity errors.

Table 9-1 Parity Errors on Slave Cassini+ Accesses

Parity Error During	**PARITY ERROR RESPONSE = 0**	**PARITY ERROR RESPONSE = 1**
Address Phase or Special Cycle	Set Detected Parity Error bit, if cycle is owned, claim and terminate normally	Set Detected Parity Error bit, do not claim the cycle. If SERR# Enable bit is set assert SERR# and set Signaled System Error bit.
Data Phase (READ)	Not applicable	Not applicable
Data Phase (WRITE)	Set Detected Parity Error bit, terminate cycle normally	Set Detected Parity Error bit, assert PERR#, terminate cycle but ignore data.

9.2.2.2 PCI Master Bus Parity Errors

Master data phase parity errors (occurring while Cassini+ DMA owns the PCI bus) are treated as fatal when PARITY ERROR RESPONSE bit is set. Otherwise data phase errors are not fatal. Address phase parity error response depends on the response of the device detecting the error. If reported via SERR# these errors are treated by Cassini+ as fatal. Table 9-2 defines Cassini+ 's response to master parity errors.

Table 9-2 Parity Errors on Master Cassini+ cycles

Parity Error During	**PARITY ERROR RESPONSE = 0**	**PARITY ERROR RESPONSE = 1**
Address Phase or Special Cycle	Cassini+ does not generate special cycles. Parity errors on Cassini+ 's DMA address phases may be reported by other agents, by activating SERR#. In this case Cassini+ stops DMA activity and interrupts CPU. Fatal Error.	
Data Phase (READ)	Set Detected Parity Error bit, and continue DMA normally	Set Detected Parity Error and Data Parity Error Detected bits. Assert PERR#, stop DMA activity and interrupt CPU. Fatal Error.
Data Phase (WRITE)	Set Detected Parity Error bit, and continue DMA normally	Set Detected Parity Error and Data Parity Error Detected bits. Stop DMA activity and interrupt CPU. Fatal Error.

10.1 Overview

The Cassini+ network interface hardware provides RAS support through a combination of features intended to increase the overall network uptime of a system using Cassini+ hardware. Cassini+ does not provide hardware fault tolerance but has mechanisms for invasive as well as non-invasive testing to allow detection of faults. Limited recovery from I/O bus parity errors improves availability over previous network products.

This chapter outlines the basic Cassini+ RAS framework, lists the specific RAS features, and concludes with areas for possible investigation for additional RAS features.

10.2 Cassini+ Features and RAS Categorization

Some features fall into one of the specific categories and others into multiple. Some general explanation of how this fits may be in order.

10.2.1 Reliability Features

Reliability results in longer periods of system operation without a failure. Cassini+ hardware is predominantly the ASIC. In order to achieve high reliability, the following items will be integral to the design, fabrication, and test processes.

1. To maintain high reliability, a Sun qualified ASIC process from a reputable vendor is used to achieve acceptable integrated circuit MTBF rates.
2. Large memories incorporate BIST testing in addition to PIO access to provide enhanced testing on these devices. All small memories without BIST have a PIO test provision. Thorough testing of large memories decrease the chances of a defective component shipped to a customer with resulting low reliability.

3. Cassini+ incorporates boundary scan via JTAG providing another means for manufacturing board test to access the internal scan chain of the chip as well as do inter-chip connection testing should Cassini+ be designed into a board product with multiple interconnected chips with boundary scan. These test features enhance board level reliability by improving the manufacturing process.

4. A variety of invasive test features within Cassini+ exist to test the ASIC and some of its surrounding components. These include various loopback test modes and test access to internal registers and memory components.

10.2.2 Serviceability Features

Features within the design which reduce the mean time to repair (MTTR) of a failed unit enhance the serviceability of the product. The test mechanisms listed above all improve the serviceability of Cassini+ in as much as they speed identification of the defective component or sub-component.

In addition to identifying the cause of a system failure, Cassini+ introduces features to allow periodic non-invasive testing to potentially detect certain failures in the ASIC or portions of the surrounding circuitry prior to them resulting in a system failure. *Should redundant cards and appropriate fail-over software be developed*, system uptime can be enhanced.

10.2.3 Availability Features

System availability is naturally improved should fail-over mechanisms be developed for a redundant Cassini+ card configuration. New with Cassini+ within the Sun network interface card family is the ability to recover from certain transient parity errors. Soft parity error reporting can be enabled warning the user of potential hardware problems. Multiple successive parity errors results in termination of the application as in current products.

10.3 Cassini+ Features for RAS Support

This is not intended to be all-inclusive but lists the primary features.

10.3.1 Memory Testing

10.3.1.1 PIO Access Testing

All Cassini+ memories have write/read access from the host.

10.3.1.2 BIST Testing

The large memories have BIST test support. BIST can be activated by the ASIC vendor during chip test. The host can also activate BIST testing through the PCI Configuration Space PCI Control Register (see Programming Model for details).

10.3.2 Scan

10.3.2.1 Internal Scan

The flip-flops in the design are connected through an internal scan chain. The ASIC vendor will test the chip through this mechanism in addition to testing with partial and full speed functional vectors.

10.3.2.2 JTAG Boundary Scan

An IEEE 1149.1 JTAG TAP controller is used to govern access to the boundary scan flip-flops, the internal scan chain, BIST activation, etc. The Cassini+ board should include a connector for manufacturing test to gain JTAG access.

10.3.3 Loopback Testing (Invasive)

Refer to the Programming Model chapter for more details.

Serialink External Loopback

External SERDES Device Loopback

MAC Internal Loopback (via MII/GMII path)

10.3.4 ASIC/System Monitoring Features

10.3.4.1 Multiplexed Control State on Local Bus

Allows control bits of selected internal state machines to be probed.

10.3.5 Physical Layer Test - MAC Registers

See Programming Model for details on the MAC registers.

10.3.5.1 FCS Error Counter

This register increments when a receive frame fails the CRC check which may be symptomatic of a flaw in the physical layer when the count becomes significant.

10.3.5.2 Alignment Error Counter

This register increments when a frame fails the CRC check and the frame contains a non-integer number of bytes.

10.3.6 Physical Layer Test/Monitor - PCS

These are affiliated with the PCS. See Programming Model for details.

Jitter Study

Link not Up (lack of Breaklink C codes from Partner)

Link not Up (due to SERDES)

Link not Up (lack of good C codes)

Link not Up (loss of sync)

Link not Up (no C codes with ACK bits set)

Link not Up (partner not sending idle)

Loss of Link (Configuration Codes)

Loss of Link (Sync Loss)

Loss of Link (loss of signal_detect)

Packets Transmitted by PCS

Packets Received by PCS (with or without error)

10.3.7 LED Monitors

Link Up

Full Duplex

TX Activity

RX Activity

Collision

10.3.8 Non-Invasive Error Detection

The driver will periodically (frequency TBD) perform checks on the hardware to test the integrity of various subsystems.

10.3.8.1 PCI Datapath Check

A 64 bit register within the slave access space of Cassini+ holds the value written into it. When s/w reads it back, it receives the value written but with both upper and lower 32 bit halves rotated to the right one bit position. (The rotation is to verify that stale information stored via charged capacitance isn't interpreted as functional datapath.)

Note that some PCI bridges support 64 bit PIOs and others only 32 bit so a full 64 bit wiring datapath check is not performed if the bridge doesn't support the transaction as a single 64 bit access.

10.3.8.2 S/W PCI Interrupt

The BIM Configuration Register (see Programming Model) allows an interrupt to be generated when the S/W interrupt bit is set.

The Datapath Check and Interrupt Check allow basic testing of the majority of the PCI connector signals.

10.3.8.3 DMA Loopback Test Packet

A small packet (size restriction TBD) can be sent periodically by the driver. The TX descriptor has a bit set indicating this to be a test packet intended to be circulated within the chip on the inside periphery of the TX and RX MACs. Also, it is not to interfere with normal traffic being received from the MAC. It is placed in a buffer just before the input to the TX MAC. When logic indicates no incoming traffic, the packet is directed to the RX logic normally receiving RX MAC data.

(Additional details to be determined include means of interrupting host when test packet is written into RX data buffer and completion ring updated, packet header fields, etc.).

An alternative to the approach stated above would be to simply put the chip into loopback mode when a long period has transpired without receipt of an incoming packet and loop back a packet as close to the physical layer as the system configuration allows. If an incoming RX packet is blocked inadvertently due to this, the upper protocol layers will provide recovery. When RX packets are being normally received, logic in the RX DMA path is behaving appropriately or at least to the level that a loopback packet would check. Also, if a protocol such as TCP be in use, the TX DMA path is being used to acknowledge the incoming RX packets. This approach is clearly simpler on the hardware design and allows testing of more circuitry closer to the physical layer than the first technique allows.

10.3.9 Limited Data Parity Error Recovery

Currently a data parity error results in PERR# assertion by either the target or master and the master reporting the error to the system. This normally results in termination of the process and a reset of the chip. The BIM Configuration Register has fields dictating how many times to retry an access with a data parity error. An interrupt can be generated when a recoverable parity error occurs for reporting/logging so that eventual service on the system is done.

10.3.9.1 Slave Write Data Parity Error

When a master writes to Cassini+ 's slave or target interface and Cassini+ detects incorrect parity on the data, Cassini+ can assert STOP# without TRDY# to signal to the master to retry the bus cycle. If the subsequent cycle occurs without the parity error the process can proceed without the application failing due to this transient parity error.

Target read cycles have parity checked by the master so this feature cannot be employed in that case. Address parity errors still result in SERR# asserted and are not recoverable.

10.3.9.2 Master Data Parity Error

When Cassini+ is the bus master, it is responsible for reporting data parity errors to the host. On read cycles Cassini+ detects the error directly. On writes, the target informs Cassini+ of the error by asserting PERR#. When data parity error recovery is enabled, Cassini+ can reissue the transaction. If the parity error was a transient the process can proceed with optional soft error reporting.

Programmer's Model 11

11.1 Address Map

11.1.1 PCI Configuration Space

Cassini+ 's PCI Configuration space follows the format of a multi-function device of the PCI Bus Specification Revision 2.2. Cassini+ 's PCI Configuration space is a read / write space accessible as bytes, half-words and words, unless indicated otherwise.

Cassini+ 's PCI Configuration space is accessible at all times. Values programmed into PCI Configuration space will not be affected by software reset, and only return to their default values upon hardware reset.

Configuration cycles for unimplemented registers in Cassini+ 's PCI Configuration space are always completed normally, with Read cycles returning all zeros, and write cycles discarding the write data.

Note: The multifunction bits in the header type registers are hardwired to 0. This prevents the OS from instantiating 4 CE drivers. In reality, there is only one device driver, serving upto 4 interrupts.

Table 11-1 Cassini+ Function #0 PCI Configuration Space

Offset	Size	R/W	Name
0x000 - 0x001	16 bit	RO	Vendor ID=0x108E
0x002 - 0x003	16 bit	RO	Device ID=0xabba
0x004 - 0x005	16 bit	R/W	Command Register
0x006 - 0x007	16 bit	R/W	Status Register
0x008	8 bit	RO	Revision ID = 0x10
0x009-0x00B	24 bit	RO	Class = 0x020000 Network Controller, Ethernet
0x00C	8 bit	R/W	Cache Line Size, in 32 bit units, resets to 0x00 (values > 128 bytes, or non power of 2 are written as zero)
0x00D	8 bit	R/W	Latency Timer
0x00E	8 bit	RO	Header Type = 0x00, single function device, standard header type
0x00F	8 bit	R/W	BIST. Resets to 0x0.
0x010 - 0x013	32 bit	R/W	Base Address Register Bits[20:0] read as zero 2 Mbyte register space is non prefetchable, mapped in 32 bit Memory Space Resets to 0x0000
0x014 - 0x02B		RO	Reserved - Read as zero
0x02C - 0x02D 0x02E - 0x02F	16 bit 16 bit	RO RO	Subsystem Vendor ID = 0x108E Subsystem ID = 0x00{value loaded from PCB}
0x030 - 0x033	32 bit	R/W	Expansion ROM Base Address, Bits [19:1] read as zero. 1 Mbyte space mapped in 32 bit Memory Space Resets to 0x0000
0x034 -0x03B		RO	Reserved - Read as zero
0x03C	8 bit	R/W	Interrupt Line
0x03D	8 bit	RO	Interrupt Pin = 0x01 => uses pin INTA#
0x03E	8 bit	RO	Min_Gnt =0x40 => 16usec bursts
0x03F	8 bit	RO	Max_Lat=0x40 => 16usec latency
0x040-0x0FF			Reserved, read as zero

Table 11-2 Cassini+ Function #1 PCI Configuration Space

0x100 - 0x101	16 bit	RO	Vendor ID=0x108E
0x102 - 0x103	16 bit	RO	Device ID=0xabbb
0x104 - 0x105	16 bit	R/W	Command Register
0x106 - 0x107	16 bit	R/W	Status Register
0x108	8 bit	RO	Revision ID = 0x10
0x109-0x10B	24 bit	RO	Class = 0x028000 Other Network Controller
0x10C	8 bit	R/W	Cache Line Size, in 32 bit units, resets to 0x00 (values > 128 bytes, or non power of 2 are written as zero)
0x10D	8 bit	R/W	Latency Timer
0x10E	8 bit	RO	Header Type = 0x00, single function device, standard header type
0x10F	8 bit	R/W	BIST. Resets to 0x0.
0x110 - 0x113	32 bit	R/W	Base Address Register Bits[20:0] read as zero 2 Mbyte register space is non prefetchable, mapped in 32 bit Memory Space Resets to 0x0000
0x114 - 0x12B		RO	Reserved - Read as zero
0x12C - 0x12D 0x12E - 0x12F	16 bit 16 bit	RO RO	Subsystem Vendor ID = 0x108E Subsystem ID = 0x00{value loaded from PCB}
0x130 - 0x133	32 bit	R/W	Expansion ROM Base Address, Bits [19:1] read as zero. 1 Mbyte space mapped in 32 bit Memory Space Resets to 0x0000
0x134 -0x13B		RO	Reserved - Read as zero
0x13C	8 bit	R/W	Interrupt Line
0x13D	8 bit	RO	Interrupt Pin = 0x02 => uses pin INTB#
0x13E	8 bit	RO	Min_Gnt =0x40 => 16usec bursts
0x13F	8 bit	RO	Max_Lat=0x40 => 16usec latency
0x140-0x1FF			Reserved, read as zero

Table 11-3 Cassini+ Function #2 PCI Configuration Space

0x200 - 0x201	16 bit	RO	Vendor ID=0x108E
0x202 - 0x203	16 bit	RO	Device ID=0xabbc
0x204 - 0x205	16 bit	R/W	Command Register
0x206 - 0x207	16 bit	R/W	Status Register
0x208	8 bit	RO	Revision ID = 0x10
0x209-0x20B	24 bit	RO	Class = 0x028000 Other Network Controller
0x20C	8 bit	R/W	Cache Line Size, in 32 bit units, resets to 0x00 (values > 128 bytes, or non power of 2 are written as zero)
0x20D	8 bit	R/W	Latency Timer
0x20E	8 bit	RO	Header Type = 0x00, single function device, standard header type
0x20F	8 bit	R/W	BIST. Resets to 0x0.
0x210 - 0x213	32 bit	R/W	Base Address Register Bits[20:0] read as zero 2 Mbyte register space is non prefetchable, mapped in 32 bit Memory Space Resets to 0x0000
0x214 - 0x22B		RO	Reserved - Read as zero
0x22C - 0x22D 0x22E - 0x22F	16 bit 16 bit	RO RO	Subsystem Vendor ID = 0x108E Subsystem ID = 0x00{value loaded from PCB}
0x230 - 0x233	32 bit	R/W	Expansion ROM Base Address, Bits [19:1] read as zero. 1 Mbyte space mapped in 32 bit Memory Space Resets to 0x0000
0x234 -0x23B		RO	Reserved - Read as zero
0x23C	8 bit	R/W	Interrupt Line
0x23D	8 bit	RO	Interrupt Pin = 0x03 => uses pin INTC#
0x23E	8 bit	RO	Min_Gnt =0x40 => 16usec bursts
0x23F	8 bit	RO	Max_Lat=0x40 => 16usec latency
0x240-0x2FF			Reserved, read as zero

Table 11-4 Cassini+ Function #3 PCI Configuration Space

0x300 - 0x301	16 bit	RO	Vendor ID=0x108E
0x302 - 0x333	16 bit	RO	Device ID=0xabbd
0x304 - 0x305	16 bit	R/W	Command Register
0x306 - 0x307	16 bit	R/W	Status Register
0x308	8 bit	RO	Revision ID = 0x10
0x309-0x30B	24 bit	RO	Class = 0x020000 Other Network Controller
0x30C	8 bit	R/W	Cache Line Size, in 32 bit units, resets to 0x00 (values > 128 bytes, or non power of 2 are written as zero)
0x30D	8 bit	R/W	Latency Timer
0x30E	8 bit	RO	Header Type = 0x00, single function device, standard header type
0x30F	8 bit	R/W	BIST. Resets to 0x0.
0x310 - 0x313	32 bit	R/W	Base Address Register Bits[20:0] read as zero 2 Mbyte register space is non prefetchable, mapped in 32 bit Memory Space Resets to 0x0000
0x314 - 0x32B		RO	Reserved - Read as zero
0x32C - 0x32D 0x32E - 0x32F	16 bit 16 bit	RO RO	Subsystem Vendor ID = 0x108E Subsystem ID = 0x00{value loaded from PCB}
0x330 - 0x333	32 bit	R/W	Expansion ROM Base Address, Bits [19:1] read as zero. 1 Mbyte space mapped in 32 bit Memory Space Resets to 0x0000
0x334 -0x33B		RO	Reserved - Read as zero
0x33C	8 bit	R/W	Interrupt Line
0x33D	8 bit	RO	Interrupt Pin = 0x04 => uses pin INTD#
0x33E	8 bit	RO	Min_Gnt =0x40 => 16usec bursts
0x33F	8 bit	RO	Max_Lat=0x40 => 16usec latency
0x340-0x3FF			Reserved, read as zero

Note – The Cache Line Register supports cache line values of 0, 16, 32, 64 and 128 bytes. For cache line sizes of zero or values other than those supported, Cassini+ will not use PCI cache command (MWI, MRL), and bursts are controlled assuming 64 bytes per cache line.

11.1.1.1 Command Register

The command register provides coarse control over a function's ability to generate and respond to PCI cycles. When a 0 is written to this register, the function is logically disconnected from the PCI bus for all accesses except configuration accesses.

Note – Please refer to §6.2.2 of the PCI bus specification for more information on the command register

Table 11-5 PCI Command Register

Bit	Usage
0	IO Space — Not implemented, read back as zero
1	Memory Space — Controls a function's response to memory space accesses: when set, allows the function to respond to memory space accesses. Reset to zero.
2	Bus Master — Controls a function's ability to act as a master on the PCI bus: when set, allows the device to behave as a bus master. Reset to zero
3	Special Cycles — Not implemented, read back as zero
4	Memory Write and Invalidate Enable — Controls whether a master can generate the Memory Write and Invalidate command (when set.) Reset to zero.
5	VGA Palette Snoop — Not implemented, read back as zero
6	Parity Error Response — This bit controls the function's response to parity errors. Parity errors are ignored when this bit is zero, and reported when this bit is one. Parity must be generated regardless of the value of this bit. Resets to zero.
7	Wait Cycle Control — Address/Data Stepping not implemented, read back as zero
8	SERR# Enable — This bit is an enable for the SERR# driver: when set, the SERR# pin driver is enabled. Resets to zero.
9	Fast Back-to-Back Enable — Not implemented, read back as zero
15 - 10	Reserved, read back as zero

11.1.1.2 Status Register

The status register is used to record information for PCI bus related events. Reads to this register behave normally; during writes, bits can only be reset, but not set. A bit is reset whenever the register is written, and the data in the corresponding bit location is a 1.

Note – Please refer to §6.2.3 of the PCI bus specification for more information on the status register

Table 11-6 PCI Status Register

Bit	Usage
0-4	Reserved - reads back as binary 10000
5	66MHz Capable — Read-only, set to 1, indicates Cassini+ is capable of running at 66MHz
6	UDF — Read-only, set to 0, indicates no User Definable Features
7	Fast Back-to-Back Capable — Read-only, set to 1, indicates Cassini+ as a target is capable of accepting fast back-to-back transactions with different agents
8	Data Parity Error Detected — Set when three conditions are met: 1) PERR# was asserted or observed asserted, 2) Cassini+ was the bus master for the transaction in which the error occurred, 3) Parity Error Response bit in Command Register is set. Resets to zero.
9-10	DEVSEL Timing — Read-only, set to 10 (slow)
11	Signaled Target Abort — When set, indicates that an illegal access to Cassini+ was terminated by Cassini+ with a target-abort. Resets to zero.
12	Received Target Abort — When set, indicates a Cassini+ bus master transaction terminated with a target-abort. Resets to zero.
13	Received Master Abort — When set, indicates a Cassini+ bus master transaction terminated with a master-abort. Resets to zero.
14	Signaled System Error — Set when Cassini+ asserts SERR#.
15	Detected Parity Error — Set when Cassini+ detects a parity error, even if the Parity Error Response in the Command Register is zero. Resets to zero.

11.1.2 Expansion ROM space

The Expansion ROM space is a 1 Mbyte read/write memory space through which byte wide external PROMs of up to 64 kbytes may be read during POST. This space is accessible as 8-bit reads. The system maps and enables this space using the Expansion ROM Base Address Register in PCI Configuration Space.

11.1.3 Cassini Register Space

This is a 2Mbyte space through which internal registers as well as the external PROM may be accessed.

This space is mapped in 32 bit memory space, and is accessible as 32-bit words. Register addresses are offset from the value written in the Base Address Register in PCI Configuration Space.

Cassini+ 's registers are not accessible in PCI IO space.

Table 11-7 Cassini+ Register Address Map

Address Offset	R/W	Default	Actual Size (bits)	Description
Global Resources				
0x0004	RW	0x0	5	Core Arbitration Weight Register
0x0008	RW	0x0	1	Infinite Burst Enable Register - Write a 1 to enable infinite burst mode. Read back as 1. - Write a 0 to disable infinite burst mode. Read back as 0.
0x000C	R-AC	0x00000000	29	Interrupt Status Register (auto-clears top level bits)
0x0010	RW	0xFFFFFFF	16	Interrupt Mask Register
0x0014	WO	0x00000000	12	Alias Clear Mask Register (used with Status Register Alias)
0x001C	RO	0x00000000	29	Interrupt Status Register Alias (selective clear)
0x1000	R-AC	0x0	3	PCI Error Status Register
0x1004	RW	0x7	3	PCI Error Status Mask Register
0x1008	RW	5'bxx000	5	BIM Configuration Register
0x100C	RW	0x00000000	32	BIM Diagnostic Register
0x1010	RW- 2 AC bits	0x00	3	Software Reset Register
0x1018	RW	undefined	0	Reserved
0x1020	RW	0x7	5	BIM Local Device Output EN Register
0x1024	RW	0x0	6	BIM Buffer Address
0x1028	RW	undefined	32	BIM Buffer Data Low Register
0x102C	RW	undefined	32	BIM Buffer Data High Register
0x1030	R - AC	0xFC	8	BIM RAM Bist Control/Status Register
0x1034	RW	0x56	16	PROBE MUX SELECT Register
0x1038	RW	0x1F	6	Interrupt Mask Register2 for INTB
0x103c	RO	0x0	5	Interrupt Status Register2 for INTB
0x1040	RW	0x0	5	Alias Clear Mask Register2 for INTB
0x1044	RO	0x0	5	Interrupt Status Register Alias2 for INTB
0x1048	RW	0x7	4	Interrupt Mask Register3 for INTC
0x104c	RO	0x0	3	Interrupt Status Register3 for INTC
0x1050	RW	0x0	3	Alias Clear Mask Register3 for INTC
0x1054	RO	0x0	3	Interrupt Status Register Alias4 for INTC

Table 11-7 Cassini+ Register Address Map

Address Offset	R/W	Default	Actual Size (bits)	Description
0x1058	RW	0x7	4	Interrupt Mask Register4 for INTD
0x105c	RO	0x0	3	Interrupt Status Register4 for INTD
0x1060	RW	0x0	3	Alias Clear Mask Register4 for INTD
0x1064	RO	0x0	3	Interrupt Status Register Alias2 for INTD
Transmit DMARegisters				
0x2004	RW	0x8C022220	32	TX Configuration Register
0x2014	RW	0x000	11	TX FIFO Write Pointer
0x2018	RW	undefined	11	TX FIFO Shadow Write Pointer
0x201C	RW	0x000	11	TX FIFO Read Pointer
0x2020	RW	undefined	11	TX FIFO Shadow Read Pointer
0x2024	RO	0x000	11	TX FIFO Packet Counter
0x2028	RO	0x40000000	32	TX State Machine Register#1
0x202C	RO	0x0	3	TX State Machine Register#2
0x2030	RO	0x00000000	32	TX Data Pointer Low
0x2034	RO	0x00000000	32	TX Data Pointer High
0x2038	RW	0x0000	13	TX Kick Register#1
0x203C	RW	0x0000	13	TX Kick Register#2
0x2040	RW	0x0000	13	TX Kick Register#3
0x2044	RW	0x0000	13	TX Kick Register#4
0x2048	RO	0x0000	13	TX Completion Register#1
0x204C	RO	0x0000	13	TX Completion Register#2
0x2050	RO	0x0000	13	TX Completion Register#3
0x2054	RO	0x0000	13	TX Completion Register#4
0x2058	RW	undefined	32	TX Completion Writeback Base Low
0x205C	RW	undefined	32	TX Completion Writeback Base High
0x2060	RW	undefined	21	TX Descriptor Base Low#1
0x2064	RW	undefined	32	TX Descriptor Base High#1
0x2068	RW	undefined	21	TX Descriptor Base Low#2
0x206C	RW	undefined	32	TX Descriptor Base High#2
0x2070	RW	undefined	21	TX Descriptor Base Low#3
0x2074	RW	undefined	32	TX Descriptor Base High#3
0x2078	RW	undefined	21	TX Descriptor Base Low#4
0x207C	RW	undefined	32	TX Descriptor Base High#4
0x2080	RW	0x0000	16	TX MaxBurst#1
0x2084	RW	0x0000	16	TX MaxBurst#2

Table 11-7 Cassini+ Register Address Map

Address Offset	R/W	Default	Actual Size (bits)	Description
0x2088	RW	0x0800	16	TX MaxBurst#3
0x208C	RW	0x0000	16	TX MaxBurst#4
0x2104	RW	undefined	11	TX FIFO Address
0x2108	RO	undefined	1	TX FIFO Tag
0x210C	RW	undefined	32	TX FIFO Data Low
0x2110	RW	undefined	32	TX FIFO Data HighT1
0x2114	RW	undefined	32	TX FIFO Data HighT0
0x2118	RO	0x090	11	TX FIFO Size
0x211C	RW	0x000	9	TX RAMBIST Control/Status
Receive DMA Registers				
0x4000	RW	0x80910	13	RX Configuration Register
0x4004	RW	0x48002002	10	RX Page Size Register
0x4008	RW	0x000	11	RX FIFO Write Pointer
0x400C	RW	0x000	11	RX FIFO Read Pointer
0x4010	RW	0x000	11	RX IPP FIFO Write Pointer
0x4014	RO	undefined	11	RX IPP FIFO Shadow Write Pointer
0x4018	RW	0x000	11	RX IPP FIFO Read Pointer
0x401C	RO	0x00000	32	RX Debug Register
0x4020	RW	0x00078	18	Rx Pause Thresholds
0x4024	RW	0x0000	13	RX Kick Register
0x4028	RW	undefined	19	RX Descriptor Ring Base Low
0x402C	RW	undefined	32	RX Descriptor Ring Base High
0x4030	RW	undefined	19	RX Completion Ring Base Low
0x4034	RW	undefined	32	RX Completion Ring Base High
0x4038	RO	0x0000	13	RX Completion Register
0x403C	RO	0x0000	15	RX Completion Head Register
0x4040	RW	0x0000	15	RX CompletionTail Register
0x4044	RW	0x00000	17	RX Blanking Register for ISR Read
0x4048	RW	0x0000	28	Rx Almost Empty Thresholds
0x404C	RW	0x0000	32	RX Random Early Detection Enable Register
0x4050	RO	0x0000	30	Rx Fifo Fullness Register
0x4054	RO	0x0000	11	Rx IPP Packet Counter Register
0x4058	RO	undefined	32	Rx Working DMA Pointer Low

Table 11-7 Cassini+ Register Address Map

Address Offset	R/W	Default	Actual Size (bits)	Description
0x405C	RO	undefined	32	Rx Working DMA Pointer High
0x4060	RW	undefined except bit 0	23	Rx BIST register
0x4064	RW	0x000	8	RX Control Fifo Write Pointer
0x4068	RW	0x000	8	RX Control Fifo Read Pointer
0x406C	RW	0x00000	17	RX Blanking Register for Alias Read
0x4080	RW	undefined	12	RX FIFO Address
0x4084	RO	undefined	1	RX FIFO Tag
0x4088	RW	undefined	32	RX FIFO Data Low
0x408C	RW	undefined	32	RX FIFO Data HighT0
0x4090	RW	undefined	32	RX FIFO Data HighT1
0x4094	RW	undefined	8	RX Control FIFO and Batching FIFO Address
0x4098	RW	undefined	32	RX Control FIFO Data Low
0x409C	RW	undefined	32	RX Control FIFO Data Mid
0x4100	RW	undefined	23	RX Control FIFO Data High and Flow_id
0x4104	RW	undefined	11	RX IPP FIFO Address
0x4108	RW	undefined	1	RX IPP FIFO Tag
0x410C	RW	undefined	32	RX IPP FIFO Data Low
0x4110	RW	undefined	32	RX IPP FIFO Data HighT0
0x4114	RW	undefined	32	RX IPP FIFO Data HighT1
0x4118	RW	undefined	32	RX Header Page Pointer Low
0x411C	RW	undefined	32	RX Header Page Pointer High
0x4120	RW	undefined	32	RX MTU Page Pointer Low
0x4124	RW	undefined	32	RX MTU Page Pointer High
0x4128	RW	undefined	11	RX Reassembly DMA Table Address
0x412C	RW	undefined	32	RX Reassembly DMA Table Data Low
0x4130	RW	undefined	32	RX Reassembly DMA Table Data Mid
0x4134	RW	undefined	15	RX Reassembly DMA Table Data High
New Registers of CASSINI+:				
0x4200	RW	undefined	19	RX Descriptor Ring 2 Base Low
0x4204	RW	undefined	32	RX Descriptor Ring 2 Base High
0x4208	RW	undefined	19	RX Completion Ring 2 Base Low
0x420c	RW	undefined	32	RX Completion Ring 2 Base High

Table 11-7 Cassini+ Register Address Map

Address Offset	R/W	Default	Actual Size (bits)	Description
0x4210	RW	undefined	19	RX Completion Ring 3 Base Low
0x4214	RW	undefined	32	RX Completion Ring 3 Base High
0x4218	RW	undefined	19	RX Completion Ring 4 Base Low
0x421c	RW	undefined	32	RX Completion Ring 4 Base High
0x4220	RW	0x0000	13	RX Kick 2 Register
0x4224	RO	0x0000	13	RX Completion 2 Register
0x4228	RO	0x0000	15	RX Completion Head 2 Register
0x422c	RW	0x0000	15	RX CompletionTail 2 Register
0x4230	RO	0x0000	15	RX Completion Head 3 Register
0x4234	RW	0x0000	15	RX CompletionTail 3 Register
0x4238	RO	0x0000	15	RX Completion Head 4 Register
0x423c	RW	0x0000	15	RX CompletionTail 4 Register
0x4240	RW	0x0000	13	Rx Almost Empty 2 Thresholds
HP Registers				
0x4140	RW	0x1651004	24	Header Parser Configuration Register
0x4144	RW	undefined	7	HP Instruction RAM Address
0x4148	RW	undefined	32	HP Instruction RAM Data Low
0x414C	RW	undefined	32	HP Instruction RAM Data Midh
0x4150	RW	undefined	32	HP Instruction RAM Data High
0x4154	RW	undefined	13	HP Data and FDB RAM Addresses
0x4158	RW	undefined	32	Header Parser Data RAM Data
0x415C	RW	undefined	32	HP Flow Database 1 Register
0x4160	RW	undefined	32	HP Flow Database 2 Register
0x4164	RW	undefined	32	HP Flow Database 3 Register
0x4168	RW	undefined	32	HP Flow Database 4 Register
0x416C	RW	undefined	32	HP Flow Database 5 Register
0x4170	RW	undefined	32	HP Flow Database 6 Register
0x4174	RW	undefined	32	HP Flow Database 7 Register
0x4178	RW	undefined	32	HP Flow Database 8 Register
0x417C	RW	undefined	32	HP Flow Database 9 Register
0x4180	RW	undefined	32	HP Flow Database 10 Register
0x4184	RW	undefined	32	HP Flow Database 11 Register
0x4188	RW	undefined	32	HP Flow Database 12 Register
0x418C	RO	undefined	32	Header Parser State Machine Register
0x4190	RO	undefined	32	Header Parser Status 1

Table 11-7 Cassini+ Register Address Map

Address Offset	R/W	Default	Actual Size (bits)	Description
0x4194	RO	undefined	32	Header Parser Status 2
0x4198	RO	undefined	32	Header Parser Status 3
0x419C	RW	undefined	32	Header Parser RAM Bist Rregister
MAC Registers				
0x6000	RW	0x0	1	TX MAC Software Reset Command
0x6004	RW	0x0	1	RX MAC Software Reset Command
0x6008	RW	0x0XXXX	17	Send Pause Command Register
0x6010	R-AC	0x000	9	TX MAC Status Register
0x6014	R-AC	0x00	7	RX MAC Status Register
0x6018	R-AC	0xXXXX0000	32	MAC Control Status Register
0x6020	RW	0x1FF	9	TX MAC Mask Register
0x6024	RW	0x7F	7	RX MAC Mask Register
0x6028	RW	0x7	3	MAC Control Mask Register
0x6030	RW	0x000	9	TX MAC Configuration Register
0x6034	RW	0x00	8	RX MAC Configuration Register
0x6038	RW	0x0	3	MAC Control Configuration Register
0x603C	RW	0x0A	7	XIF Configuration Register
0x6040	RW	undefined	8	InterPacketGap0 Register
0x6044	RW	undefined	8	InterPacketGap1 Register
0x6048	RW	undefined	8	InterPacketGap2 Register
0x604C	RW	undefined	10	SlotTime Register
0x6050	RW	undefined	10	MinFrameSizeRegister
0x6054	RW	undefined	15	MaxFrameSize Register
0x6058	RW	undefined	10	PA Size Register
0x605C	RW	undefined	4	JamSize Register
0x6060	RW	undefined	8	Attempt Limit Register
0x6064	RW	undefined	16	MAC Control Type Register
0x6080	RW	undefined	16	MAC Address 0 Register
0x6084	RW	undefined	16	MAC Address 1 Register
0x6088	RW	undefined	16	MAC Address 2 Register
0x608C	RW	undefined	16	MAC Address 3 Register
0x6090	RW	undefined	16	MAC Address 4 Register
0x6094	RW	undefined	16	MAC Address 5 Register
0x6098	RW	undefined	16	MAC Address 6 Register
0x609C	RW	undefined	16	MAC Address 7 Register

Table 11-7 Cassini+ Register Address Map

Address Offset	R/W	Default	Actual Size (bits)	Description
0x60A0	RW	undefined	16	MAC Address 8 Register
0x60A4	RW	undefined	16	MAC Address 9 Register
0x60A8	RW	undefined	16	MAC Address 10 Register
0x60AC	RW	undefined	16	MAC Address 11 Register
0x60B0	RW	undefined	16	MAC Address 12 Register
0x60B4	RW	undefined	16	MAC Address 13 Register
0x60B8	RW	undefined	16	MAC Address 14 Register
0x60BC	RW	undefined	16	MAC Address 15 Register
0x60C0	RW	undefined	16	MAC Address 16 Register
0x60C4	RW	undefined	16	MAC Address 17 Register
0x60C8	RW	undefined	16	MAC Address 18 Register
0x60CC	RW	undefined	16	MAC Address 19 Register
0x60D0	RW	undefined	16	MAC Address 20 Register
0x60D4	RW	undefined	16	MAC Address 21 Register
0x60D8	RW	undefined	16	MAC Address 22 Register
0x60DC	RW	undefined	16	MAC Address 23 Register
0x60E0	RW	undefined	16	MAC Address 24 Register
0x60E4	RW	undefined	16	MAC Address 25 Register
0x60E8	RW	undefined	16	MAC Address 26 Register
0x60EC	RW	undefined	16	MAC Address 27 Register
0x60F0	RW	undefined	16	MAC Address 28 Register
0x60F4	RW	undefined	16	MAC Address 29 Register
0x60F8	RW	undefined	16	MAC Address 30 Register
0x60FC	RW	undefined	16	MAC Address 31 Register
0x6100	RW	undefined	16	MAC Address 32 Register
0x6104	RW	undefined	16	MAC Address 33 Register
0x6108	RW	undefined	16	MAC Address 34 Register
0x610C	RW	undefined	16	MAC Address 35 Register
0x6110	RW	undefined	16	MAC Address 36 Register
0x6114	RW	undefined	16	MAC Address 37 Register
0x6118	RW	undefined	16	MAC Address 38 Register
0x611C	RW	undefined	16	MAC Address 39 Register
0x6120	RW	undefined	16	MAC Address 40 Register
0x6124	RW	undefined	16	MAC Address 41 Register
0x6128	RW	undefined	16	MAC Address 42 Register

Table 11-7 Cassini+ Register Address Map

Address Offset	R/W	Default	Actual Size (bits)	Description
0x612C	RW	undefined	16	MAC Address 43 Register
0x6130	RW	undefined	16	MAC Address 44 Register
0x6134 to 0x6148 UNUSED				
0x614C	RW	undefined	16	Address Filter 0 Register
0x6150	RW	undefined	16	Address Filter 1 Register
0x6154	RW	undefined	16	Address Filter 2 Register
0x6158	RW	undefined	8	Address Filter 2&1 Mask Register
0x615C	RW	undefined	16	Address Filter 0 Mask Register
0x6160	RW	undefined	16	Hash Table 0 Register
0x6164	RW	undefined	16	Hash Table 1 Register
0x6168	RW	undefined	16	Hash Table 2 Register
0x616C	RW	undefined	16	Hash Table 3 Register
0x6170	RW	undefined	16	Hash Table 4 Register
0x6174	RW	undefined	16	Hash Table 5 Register
0x6178	RW	undefined	16	Hash Table 6 Register
0x617C	RW	undefined	16	Hash Table 7 Register
0x6180	RW	undefined	16	Hash Table 8 Register
0x6184	RW	undefined	16	Hash Table 9 Register
0x6188	RW	undefined	16	Hash Table 10 Register
0x618C	RW	undefined	16	Hash Table 11 Register
0x6190	RW	undefined	16	Hash Table 12 Register
0x6194	RW	undefined	16	Hash Table 13 Register
0x6198	RW	undefined	16	Hash Table 14 Register
0x619C	RW	undefined	16	Hash Table 15 Register
0x61A0	RW	undefined	16	Normal Collision Counter
0x61A4	RW	undefined	16	First Attempt Successful Collision Counter
0x61A8	RW	undefined	16	Excessive Collision Counter
0x61AC	RW	undefined	16	Late Collision Counter
0x61B0	RW	undefined	16	Defer Timer
0x61B4	R-AC	undefined	8	Peak Attempts Register
0x61B8	RW	undefined	16	Receive Frame Counter
0x61BC	RW	undefined	16	Length Error Counter
0x61C0	RW	undefined	16	Alignment Error Counter
0x61C4	RW	undefined	16	FCS Error Counter
0x61C8	RW	undefined	16	RX code Violation Error Counter

Table 11-7 Cassini+ Register Address Map

Address Offset	R/W	Default	Actual Size (bits)	Description
0x61CC	RW	undefined	10	Random Number Seed Register
0x61D0	RO	0x00	27	State Machine Register
MIF Registers				
0x6200	RW	0x0	1	MIF Bit-Bang Clock
0x6204	RW	0x0	1	MIF Bit-Bang Data
0x6208	RW	0x0	1	MIF Bit-Bang Output Enable
0x620C	RW		32	MIF Frame/Output Register
0x6210	RW		15	MIF Configuration Register
0x6214	RW	0xFFFF	16	MIF Mask Register
0x6218	R-AC		32	MIF Status Register
0x621C	RO		7	MIF State Machine Register
PCS/Serialink				
0x9000	RW	0x1000	5	PCS MII Control Register
0x9004	RO	0x0608	16	PCS MII Status Register
0x9008	RW	0x01e0	16	PCS MII Advertisement Register
0x900C	RW	undefined	16	PCS MII Link Partner Ability Register
0x9010	RW	0x0	6	PCS Configuration Register
0x9014	RW	0x0	27	PCS State Machine and Diagnostic Register
0x9018	R-AC	0x0	1	PCS Interrupt Status Register
0x9050	RW	0xX	2	Datapath Mode Register
0x9054	RW	0x000	3	Serdes Control Register
0x9058	RW	0x0	3	Shared Output Select Register
0x905C	RO	0x0	2	Serdes State Register
0x9060	RO	0x0	22	PCS Packet Counter Register
LocalBus Devices				
0x100000 - 0x17FFFF	R/W			Expansion ROM run time access
0x180000 - 0x1FFFFF	R/W			Secondary local bus device

11.1.4 Register Description

11.1.4.1 Global Resources

Core Arbitration Weight Register (RW)

This register is used to set the weights for the weighted round robin arbiter. For example, when Rx weight is set to one and Tx weight set to zero, the Rx will be granted twice the transfer credit as Tx for its next turn to access the PCI bus.

Table 11-8 Core Arbitration Weight Register

Bits	Field Name	Description
[1:0]	Rx DMA Weight	Determines the multiplication factor for granting credit to the Rx side during the Weighted Round Robin Arbitration. 2'b00 : multiply by 1 2'b01 : multiply by 2 2'b10 : multiply by 4 2'b11 : multiply by 8
[3:2]	Tx DMA Weight	Determines the multiplication factor for granting credit to the Tx side during the Weighted Round Robin Arbitration. 2'b00 : multiply by 1 2'b01 : multiply by 2 2'b10 : multiply by 4 2'b11 : multiply by 8
[4]	Weighted Round Robin Disable	When set to '1' the Weighted Round Robin Arbiter is disabled and packets are arbitrated on packet-boundaries in a simple round robin fashion.

Default: 0x000

Core Infinite Burst Enable Register (RW)

This register is used to turn on or off infinite burst. Cacheline sized bursts will be used instead (as programmed into the slave configuration space).

Table 11-9 Core Infinite Burst Enable Register

Bits	Field Name	Description
[0]	Infinite Burst Enable	Allows BIM to send bursts across the PCI bus which are longer than the cacheline size. The burst sizes will be determined by the length of the packet or descriptor transfer and the maximum length allowed by the target.

Default: 0x000

Interrupt Status Register (R-AC)

This is the top level register used to communicate to the software events that were detected by the hardware. If a status bit is set to '1', it indicates that the corresponding event has occurred. Top level interrupts (stored in bits 0 through 9 of the register) are automatically cleared to '0' when the Status Register is read. Second level interrupts (reported in bits 13 through 18 of the register) are cleared at the source (for example, by reading the corresponding status register). The value of the TX Completion Register 3 is replicated in bits 19 through 31 of this register.

Table 11-10 Interrupt Status Register

Bits	Field Name	Description
[0]	TX_INT_ME	This interrupt is set when a frame with the INT ME descriptor bit set is completely transferred from the host queue to the TX FIFO.
[1]	TX_ALL	This interrupt is set when all transmit frames were transferred into the TX FIFO. Namely, the TX Kick Register and TX Completion Register values coincide. By setting the PACED_MODE it is also possible to restrict this interrupt to all transmit frames transferred AND the TX FIFO being empty.
[2]	TX_DONE	Set when any frame is transferred into the TX FIFO.
[3]	TX_TAG_ERROR	Set when the TX FIFO tag framing is corrupted, namely anything other than two consecutive tag bits set. This is a fatal error.
[4]	RX_DONE	At least one frame was transferred from the RX FIFO to host memory. Set when the RX Completion Register is updated. May be delayed by the receive interrupt blanking.
[5]	Rx_Buffer_Not_Available	Set when there are no free receive buffers. Namely, the RX Kick Register and RX Completion Register values coincide.
[6]	RX_TAG_ERROR	Set when the RX FIFO tag framing is corrupted, namely anything other than two consecutive tag bits set. This is a fatal error.
[7]	RX_COMP_FULL	Set when there is no more room in the completion ring to post descriptors. Namely, when the RX Completion Head pointer increments to almost reach the RX Completion Tail Register.
[8]	RX_BUF_ALMOST_EMPTY	Set when there is less than the programmable threshold number of free descriptors available for hardware use.
[9]	RX_COMPLETION_ALMOST_FULL	Set when there is less than the programmable threshold number of descriptors spaces available for hardware use in the completion descriptor ring.

Table 11-10 Interrupt Status Register

Bits	Field Name	Description
[10]	RX_LENGTH_MISMATCH	Set when the length field from the MAC does not match the length of the non-reassembly packet unloaded from the fifo during DMA or when the Header Parser provides a TCP header and payload size which is larger than the packet size provided by the MAC. This is a fatal error.
[11]		
[12]	Summary Interrupt Bit	If an interrupt is generated onto the PCI bus, this bit will be set. It clears when the interrupt status register is read or when selectively cleared upon reading of the status alias register. It's purpose is to leave a flag for the device driver's interrupt service routine when the interrupt polling thread reads and services interrupts. This way, the driver can claim the interrupt even if it does not service one.
[13]	PCS_INT	The PCS Interrupt Status Register is indicating an interrupt condition.
[14]	TX _MAC_INT	The Status Register in the TX MAC has at least one unmasked interrupt set.
[15]	RX_MAC_INT	The Status Register in the RX MAC has at least one unmasked interrupt set.
[16]	MAC_CTRL_INT	The Status Register in the MAC Control has at least one unmasked interrupt set.
[17]	MIF_Interrupt	The Status Register in the MIF has at least one unmasked interrupt set.
[18]	PCI_ERROR_INT	The PCI Error Status Register in the BIF has at least one unmasked interrupt set.
[31:19]	Tx Completion Reg 3	Same as value of Tx Completion Reg 3, copied here to help driver fetch completion register value for the common case.

Interrupt Mask Register (RW)

This register is used to determine which status events will cause an interrupt. If a mask bit is cleared to '0', the corresponding event causes an interrupt signal to be generated. The layout of this register corresponds bit-by-bit to the layout of the interrupt bits in the Status Register

Default: All implemented bits default high.

Alias Clear Mask Register (RW)

This location may be used to specify which top level interrupt bits will be cleared during read of the Status Register Alias. Its layout corresponds to the layout of the top level interrupt bits of the Interrupt Status Register. Bit positions written high will be cleared, while bit positions written low will not be cleared when the Status Register Alias is read. This register is to assist in claiming of interrupts when software uses a polling method for interrupts.

Default: All implemented bits default low.

Status Register Alias (RO)

This location presents the same view as the Global Interrupt Status Register, except that reading from this location clears its bits selectively based on the Selective Clear Mask Register.

PCI Error Status Register (R-AC)

This register indicates PCI Errors have occurred.

Table 11-11 PCI Error Status Register

Bits	Field Name	Description
[0]	Reserved	
[1]	DTRTO	Delayed transaction timeout. Set if no read retry after 2^15 clocks.
[2]	OTHER	Set by other PCI Errors. the specific error may be read from the PCI Status Register in PCI Configuration space.
[3]	dmaw_zero_cnt_err_sy	BIM received a zero count DMA write request
[4]	dmar_zero_cnt_err_sy	BIM received a zero count DMA read request
[5]	retry count timout	BIM received 255 retries in a row during a DMA

PCI Error Mask Register (RW)

This register is used to determine which PCI Error status events will set the PCI_ERROR_INT in the Interrupt Status Register. If a mask bit is cleared to '0', the corresponding event causes an interrupt signal to be generated. The layout of this register corresponds bit-by-bit to the layout of the PCI Error Status Register

Default: All implemented bits default high.

BIM Configuration Register (RW)

This register is used to configure PCI related parameters that are not in PCI Configuration space. These values convey specific system information for the BIM block to optimize its performance. Default values indicate no special knowledge is assumed by the BIM. M66EN is a read only bit.

Table 11-12 BIM Configuration Register

Bits	Field Name	Description
[0]	Reserved	
[1]	Reserved	
[2]	bd64_dis	R/W bit. Set to 1 to disable 64-bit mode

Table 11-12 BIM Configuration Register

Bits	Field Name	Description
[3]	m66en	Read only bit. State of external M66EN pin 1 = Cassini+ PCI clock is 66 Mhz 0 = Cassini+ PCI clock is less than 66 Mhz
[4]	bus32wide	Read only bit. 1 = PCI slot is 32-bit 0 = PCI slot is 64-bit
[5]	DPAR_INTR_ENABLE	Detected Parity Error Interrupt Enable
[6]	RMA_INT_EN	Master Abort interrupt enable bit
[7]	RTA_INT_EN	Target Abort interrupt enable bit
[8]	reserved	
[9]	disable_bim (RW)	This bit is used to stop all further PCI dma transactions issued by the BIM. This bit allows device driver to quiesce the BIM before issuing a global reset
[10]	bim_disabled (RO)	This bit indicates that the bim dma has been suspended
[11]	block_perr	When set, this bit will block PERR# to the PCI bus. Default as 0.

BIM Diagnostic Register (RW)

Table 11-13 BIM Diagnostic Register

Bits	Field Name	Description
[31:30]	Reserved	
[29:8]	mstr_SM[21:0]	PCI Master Controller state machine bits
[7]	Reserved	
[6:0]]	brst_SM[6:0]	PCI Burst Controller state machine bits

Software Reset Register (RW-AC)

The lowest two bits of this register are used to perform an Individual Software Reset to the TX or RX functions (when the corresponding bit is set), a Global Software Reset to the Cassini+ (when both bits are set). These bits can be set to

'1' using a Programmed IO write to the defined address. They become "self-cleared" after the corresponding reset command has been executed. The third bit(RSTOUT) is not self clearing and is used to activate the RSTOUT# pin. The

Table 11-14 Software Reset Register

Bits	Field Name	Description
[0] .	TX Software Reset	Writing a1 will reset the TXDMAengine. Software has to poll this register until it is ckeared to 0
[1]	RX Software Reset	Writing a1 will reset the RXDMA engine. Software must poll this register until it is cleared to 0 A core software reset will be issued when software write both bit0 and bit2 simultateously. **Note: after issuing a software reset, software must wait at least 3ms before polling this register.**
[2]	RSTOUT	When set this bit forces the RSTOUT# pin active (low). When clear, RSTOUT# follows the level of the PCI reset input pin (RST#). This bit can be used to reset the PHY and any other circuitry connected to the RSTOUT# pin. default to 0
[3]	Block_reset_to_pcs_slink	When this bit along with bits 1 and 0 are set, a global reset is issued to all modules except the pcs and slink modules. This allows device driver to reset the BIM , tx and rx dma channels and the MAC without bringing the link down.
[7:3]	Reserved	
[14:8]	breq_sm_[6:0]	breq_SM[6:0]
[15]	Reserved	
{18:16]	c_pciarb_st	000:ARB_IDLE1 001:ARB_IDLE2 010:ARB_WB_ACK 011:ARB_WB_WAT 100:ARB_RB_ACK 101:ARB_RB_WAT 110:ARB_RB_END 111:ARB_WB_END
[19]	Reserved	
[21:20]	c_rdpci_st[1:0]	00:RD_PCI_WAT 01:RD_PCI_RDY 11:RD_PCI_ACK
[23:22]	c_rdarb_st[1:0]	00: AD_IDL_RX 01: AD_ACK_RX 10: AD_ACK_TX 11: AD_IDL_TX
[24]	Reserved	
[26:25]	c_wrpci_st[1:0]	00: WR_PCI_WAT 01: WR_PCI_RDY 11: WR_PCI_ACK

Table 11-14 Software Reset Register

Bits	Field Name	Description
[29:27]	c_wrarb_st[2:0]	000: ARB_IDLE1 001: ARB_IDLE2 010: ARB_TXACK 011: ARB_TX_WT 100: ARB_RXACK 110: ARB_RX_WT
[31:30]	Reserved	

RSTOUT# pin is also activated by the local PCI reset driven by the hot swap logic and is only tri-stateable with iddq_.

Programming Restrictions: *To ensure proper operation of the hardware after a Software Reset (Individual or Global), this register must be polled by the software. When both bits read back as 0's, the software is allowed to continue to program the hardware.*

BIM RAM Bist Control/Status

There is one bist controllers in bim area. The bist controller for the read buffer is running on pci clock. Setting bit 0 of this register to 1 starts the bist operation on read buffer. This bit is automatically cleared when the operation is done.

Bits	Field Name	Description
[0]	Bist start on read buffer	Start bist operation for bim read buffer when set
[1]	Reserved	
[2]	Bist pass summary for read buffer	Summarize bist pass status for bim read buffer
[3]	Reserved	
[4]	Read buffer low bank pass	Low bank of read buffer passes bist operations
[5]	Read buffer high bank pass	high bank of read buffer passes bist operations
[6]	Reserved	
[7]	Reserved	

BIM Local Device Output Enable Register (RW)

For external programmability to the Local Bus devices, specific output enables are provided for each device's chip select and an output enable for the rest of the outputs from Cassini+ to the devices.

Bits	Field Name	Description
[0]	lb_pad_tri_l	Address bus, RW signal and OE signal output enable on the Local Bus Interface. These are shared signals between both local bus devices. When set to '0' the signals are tristated.
[1]	lb_pcs_tri_l	PROM chip select output enable. When set to '0' the signal is tristated.
[2]	lb_ext_tri_l	Secondary local bus device chip select output enable. When set to '0' the signal is tristated.
[3]	soft_bit_0	software programmable control bit 0
[4]	soft_bit_1	software programmable control bit 1
[5]	Internal hardware reset	when set to 1, this bit is used to generate an internal hardware reset to the most of the chip. However, the bus32wide flag is kept intact to retain the last configuration setting. All pci configuration setting must be re-initialized.

Defaults to 0x7.

Two software programmable bits are connected directly to the asic pins. These two bits serve as general purpose control/status bits.

BIM Buffer Address (RW)

For diagnostic purposes a PIO path has been provided into the BIM Read and Write Buffers. This 6-bit register holds the address of the 24 entry Read and Write Buffers.

Bits	Field Name	Description
[5:0]	Buffer Address	There are 24 entries each in the Read and Write buffers.
[6]	Read/Write Buffer Select	Read Buffer access = 0 Write Buffer access = 1

Defaults to undefined.

BIM Buffer Data Low, Data High (RW)

These registers are used for diagnostics access to any BIM Buffer location. Every access involves 64 bits. The 32 least significant bits are accessed through the BIM Buffer Data Low, the 32 most significant bits through the BIM Buffer Data High register. A write sequence to the BIM Read Buffer consists of:

- Writing the BIM Buffer Address Register with a value of 0 to 23.

- Writing the BIM Buffer Data Low Register.
- Writing the BIM Buffer Data High Register.

A read sequence to the BIM Read Buffer consists of:

- Writing into the BIM Buffer Address Register
- Reading from the BIM Buffer Data Low and Data High Registers.

Defaults to undefined.

Programming Restrictions: *The Data High should be the last access of the sequence.*

BIM diagnostic probe mux select register (RW)

This register is used to enable the probe monitoring mode and to select the data appearing on the P_A* bus.

Bits	Field Name	Description
[31]	Probe enable	Setting this bit to one will allow probe signals to be driven onto the local bus P_A[15:0] for debugging
[31:16]	reserved	
[15:8]	sub_mux_sel[7:0]	1: select txdma_wr address and size

[7:4]	probe_sel_hi[3:0]	Used to select which module to appear on P_A[15:8]: 4'h0: internal_probe[7:0] 4'h1: internal_probe[15:8] 4'h2: internal_probe[23:16] 4'h3: internal_probe[31:24] 4'h4: pci_io_probe[7:0] 4'h5: pci_io_probe[15:8]; 4'h6: pci_io_probe[23:16]; 4'h7: pci_io_probe[31:24]; 4'h8: pci_io_probe[39:32]; 4'h9: pci_io_probe[47:40]; 4'ha: pci_io_probe[55:48]; 4'hb: pci_io_probe[63:56]; 4'hc: rx_probe[7:0]; 4'hd: tx_probe[7:0]; 4'he: hp_probe[7:0]; 4'hf: mac_probe[7:0];
[3:0]	probe_sel_lo[3:0]	Used to select which module to appear on P_A[7:0]: 4'h0: internal_probe[7:0] 4'h1: internal_probe[15:8] 4'h2: internal_probe[23:16] 4'h3: internal_probe[31:24] 4'h4: pci_io_probe[7:0] 4'h5: pci_io_probe[15:8]; 4'h6: pci_io_probe[23:16]; 4'h7: pci_io_probe[31:24]; 4'h8: pci_io_probe[39:32]; 4'h9: pci_io_probe[47:40]; 4'ha: pci_io_probe[55:48]; 4'hb: pci_io_probe[63:56]; 4'hc: rx_probe[7:0]; 4'hd: tx_probe[7:0]; 4'he: hp_probe[7:0]; 4'hf: mac_probe[7:0];
		internal_probe[31:24] = {rxdma_req, rxdma_ack, rxdma_rdy,c_wrarb_st,c_wrpci_st} **internal_probe[23:16]** = {txdma_rd_req, txdma_rd_ack, txdma_rd_rdy,rxdma_rd, c_rdarb_st,c_rdpci_st } **internal_probe[15:8]** = {pci_wbuf_comp, pci_wpkt_comp, pci_rbuf_comp, pci_rpkt_comp,txdma_wr_req, txdma_wr_ack, txdma_wr_rdy,txdma_wr_xfr_done} **internal_probe[7:0]** = {c_pciarb_st,wtc_empty_w,wtc_full_w,wtc_empty_w,wtc_empty_r, post_pci_}

Interrupt Status Register 2 (R-AC)

This ISR2 register is associated with pci interrupt B. It provides the status of the second descriptor ring and completion ring 2.

Table 11-15 Interrupt Status Register2 for INTB#

Bits	Field Name	Description
[0]	RX_DONE	At least one frame was transferred from the RX FIFO to host memory. Set when the RX Completion Register is updated. May be delayed by the receive interrupt blanking.
[1]	RX_COMP_FULL	Set when there is no more room in the completion ring to post descriptors. Namely, when the RX Completion Head pointer increments to almost reach the RX Completion Tail Register.
[2]	RX_COMPLETION_ALMOST_FULL	Set when there is less than the programmable threshold number of descriptors spaces available for hardware use in the completion descriptor ring.
[3]	Rx_Buffer_Not_Available	Set when there are no free receive buffers. Namely, the RX Kick Register and RX Completion Register values coincide.
[4]	RX_BUF_ALMOST_EMPTY	Set when there is less than the programmable threshold number of free descriptors available for hardware use.

Interrupt Mask Register 2 (RW)

This register is used to determine which status events will cause an interrupt. If a mask bit is cleared to '0', the corresponding event causes an interrupt signal to be generated. The layout of this register corresponds bit-by-bit to the layout of the interrupt bits in the Status Register

Table 11-16 Interrupt mask register 2

Bits	Field Name	Description
[0]	RX_DONE mask	mask bit
[1]	RX_COMP_FULL mask	mask bit
[2]	RX_COMPLETION_ALMOST_FULL mask	mask bit
[3]	Rx_Buffer_Not_Available mask	mask bit

Table 11-16 Interrupt mask register 2

Bits	Field Name	Description
[4]	RX_BUF_ALMOST_EMPTY mask	mask bit
[5:6]	reserved	
[7]	intb_enable	When set, intb will be generated when one of the flags are set. This bit is also used to enable the 2nd descriptor ring

Default: 0x1F.

Alias Clear Mask Register 2 (RW)

This location may be used to specify which top level interrupt bits will be cleared during read of the Status Register Alias. Its layout corresponds to the layout of the interrupt bits of the Interrupt Status Register 2. Bit positions written high will be cleared, while bit positions written low will not be cleared when the Status Register Alias is read. This register is to assist in claiming of interrupts when software uses a polling method for interrupts.

Default: 0x0.

Status Register Alias2 (RO)

This location presents the same view as the Interrupt Status Register 2, except that reading from this location clears its bits selectively based on the Selective Clear Mask Register.

Interrupt Status Register 3 (R-AC)

This ISR3 register is associated with pci interrupt C. It provides the status of the third completion ring.

Table 11-17 Interrupt Status Register3 for INTC#

Bits	Field Name	Description
[0]	RX_DONE	At least one frame was transferred from the RX FIFO to host memory. Set when the RX Completion Register is updated. May be delayed by the receive interrupt blanking.
[1]	RX_COMP_FULL	Set when there is no more room in the completion ring to post descriptors. Namely, when the RX Completion Head pointer increments to almost reach the RX Completion Tail Register.
[2]	RX_COMPLETION_ALMOST_FULL	Set when there is less than the programmable threshold number of descriptors spaces available for hardware use in the completion descriptor ring.

Interrupt Mask Register 3 (RW)

This register is used to determine which status events will cause an interrupt. If a mask bit is cleared to '0', the corresponding event causes an interrupt signal to be generated. The layout of this register corresponds bit-by-bit to the layout of the interrupt bits in the Status Register

Table 11-18 Interrupt Mask register 3

Bits	Field Name	Description
[0]	RX_DONE mask	mask bit
[1]	RX_COMP_FULL mask	mask bit
[2]	RX_COMPLETION_ALMOST_FULL mask	mask bit
[3:6]	reserved	
[7]	intc_enable	When set, intc will be generated when one of the flags are set. This bit is also used to enable the 3nd completion descriptor ring

Default: 0x7.

Alias Clear Mask Register 3 (RW)

This location may be used to specify which top level interrupt bits will be cleared during read of the Status Register Alias. Its layout corresponds to the layout of the top level interrupt bits of the Interrupt Status Register. Bit positions written high will be cleared, while bit positions written low will not be cleared when the Status Register Alias is read. This register is to assist in claiming of interrupts when software uses a polling method for interrupts.

Default: All implemented bits default low.

Status Register Alias3 (RO)

This location presents the same view as the Global Interrupt Status Register, except that reading from this location clears its bits selectively based on the Selective Clear Mask Register.

Interrupt Status Register 4 (R-AC)

This ISR4 register is associated with pci interrupt C. It provides the status of the 4th completion ring.

Table 11-19 Interrupt Status Register2 for INTD#

Bits	Field Name	Description
[0]	RX_DONE	At least one frame was transferred from the RX FIFO to host memory. Set when the RX Completion Register is updated. May be delayed by the receive interrupt blanking.
[1]	RX_COMP_FULL	Set when there is no more room in the completion ring to post descriptors. Namely, when the RX Completion Head pointer increments to almost reach the RX Completion Tail Register.
[2]	RX_COMPLETION_ALMOST_FULL	Set when there is less than the programmable threshold number of descriptors spaces available for hardware use in the completion descriptor ring.

Interrupt Mask Register 4 (RW)

This register is used to determine which status events will cause an interrupt. If a mask bit is cleared to '0', the corresponding event causes an interrupt signal to be generated. The layout of this register corresponds bit-by-bit to the layout of the interrupt bits in the Status Register

Table 11-20 Interrupt Mask register 4

Bits	Field Name	Description
[0]	RX_DONE mask	mask bit
[1]	RX_COMP_FULL mask	mask bit
[2]	RX_COMPLETION_ALMOST_FULL mask	mask bit
[3:6]	reserved	
[7]	intd_enable	When set, intd will be generated when one of the flags are set. This bit is also used to enable the 4th completion descriptor ring

Default: 0x7.

Alias Clear Mask Register 4 (RW)

This location may be used to specify which top level interrupt bits will be cleared during read of the Status Register Alias. Its layout corresponds to the layout of the top level interrupt bits of the Interrupt Status Register. Bit

positions written high will be cleared, while bit positions written low will not be cleared when the Status Register Alias is read. This register is to assist in claiming of interrupts when software uses a polling method for interrupts.

Default: All implemented bits default low.

Status Register Alias 4 (RO)

This location presents the same view as the Interrupt Status Register, except that reading from this location clears its bits selectively based on the Selective Clear Mask Register.

11.1.5 TX DMA Programmable Resources

TX Kick Register#1, #2 ,#3 ,#4 (RW)

This 13-bit register is written by the host CPU with the descriptor value that follows the last valid transmit descriptor. The TX Kick Register#i and TX Completion Register#i values are used by the transmit DMA engine to control TX descriptor fetching. If the value indicates that more than one valid TX descriptor is available within the cache line being read, Cassini will internally cache up to four of them. Initialized to zero on reset.

TX Completion Register#1 ,#2 ,#3 ,#4 (RO)

This 13-bit register stores the descriptor value that follows the last descriptor already processed by Cassini. In addition to the internal use of this value for descriptor fetching, the host CPU may read this register to determine which transmit resources may be reclaimed. Descriptors up to but excluding the register value may be re-claimed. The DMA state machine updates the value of this register and writes back this value into host memory space once the transmit data buffer contents are entirely loaded into the TX FIFO, but before setting any of the pertinent bits in the Interrupt Status Register (TX_INT_ME, TX_ALL, TX_DONE). In cases where a transmit frame has multiple descriptors this register is updated on descriptor boundaries. Initialized to zero on reset.

TX Configuration Register (RW)

This register stores parameters that control the operation of the transmit DMA channel.

Table 11-21 TX Configuration Register

Bits	Field Name	Description
[0]	Tx_DMA_Enable	When set to '1', the TX DMA channel is enabled. When cleared to '0', the TX DMA operation will orderly stop as soon as the transfer of the current data buffer has been completed. Defaults low.
[1]	Tx_FIFO_PIO_Sel	When set to '1', the TX DMA FIFO can be accessed with PIOs using the TX FIFO address and data registers. It is recommended that the TX DMA be disabled before performing the TX FIFO PIOs in case the intent is to retain the integrity of the TX FIFO for continued transmission of packet data.
[5:2] [9:6] [13:10] [17:14]	Tx_Desc_Ring_Size#1 Tx_Desc_Ring_Size#2 Tx_Desc_Ring_Size#3 Tx_Desc_Ring_Size#4	This field determines the number of descriptor entries in the Tx ring#i. Default: ox8; 8k descriptor entries.
[19:18]	Reserved	
[20]	Paced_Mode	When set to '1', the TX_ALL interrupt (bit 1 in the Status Register) will become set only after the TX FIFO becomes empty. If cleared to '0', the TX_ALL interrupt is only a function of the descriptor queue being empty. Defaults low.
[21:23]	Reserved	Do not map to any internal logic.
[24]	Txdma_rdpipe_disable	This bit should always be set to a '1'.
[25]	CompWb_PerPkt_Q1_En	When set to '1', completion writeback happens at the end of every packet kicked through Q1.
[26]	CompWb_PerPkt_Q2_En	When set to '1', completion writeback happens at the end of every packet kicked through Q2.
[27]	CompWb_PerPkt_Q3_En	When set to '1', completion writeback happens at the end of every packet kicked through Q3.
[28]	CompWb_PerPkt_Q4_En	When set to '1', completion writeback happens at the end of every packet kicked through Q4.
[29]	PreIntr_Compb_disable	When set to '1', the pre-interrupt completion writeback is disabled.
[30:31]	Ctx_diag_sel	When set to '00' the tx test ports are connected to tx_mac_req, tx_mac_retry_req, tx_ack and tx_tag. When set to '01' the tx test ports are connected to txdma_rd_req, txdma_rd_ack, txdma_rd_rdy and txdma_rd_type0. When set to '11' the tx test ports are connected to txdma_wr_req, txdma_wr_ack, txdma_wr_rdy and txdma_wr_xfr_done.

Default: 3F000001

*Table 11-22*TX Descriptor Ring Size values

Tx_Desc_Ring_Size#i	Ring Size
0	32
1	64
2	128
3	256
4	512
5	1k
6	2k
7	4k
8	8k
9-15	Reserved

TX Descriptor Base Low#1, #2, #3, #4 and High#1, #2, #3, #4 (RW)

The 53 most significant bits of this register are used as the base address for the TX descriptor ring. The 11 least significant bits are not stored, and assumed to be zero.

Programming Restrictions: *The Transmit Descriptor Pointer must be initialized to a 2KByte-aligned value after power-on or software reset.*

TX Completion Writeback Base Low and High (RW)

The value of the Tx Completion registers #1 through #4 are written into the host memory space with the base address from these two 32bit registers. Each of the completion register occupies 2 bytes of host memory space, and are written contiguously from the base address with Tx completion register#1 first followed by Tx completion register#2, Tx completion register#3, and finally Tx completion register#4. A total of 8 bytes of hostmemory space must be allocated for this purpose. The Completion WriteBack address must be 8 byte aligned.

Table 11-23 Completion Register WriteBack Array layout.

Offset from Completion WriteBack base Address	Completion# byte
0	Completion1_MSB
1	Completion1_LSB
2	Completion2_MSB
3	Completion2_LSB
4	Completion3_MSB
5	Completion3_LSB
6	Completion4_MSB
7	Completion4_LSB

Programming Note – The completion register values are written back to the host memory only prior to tx_int_me and tx_all interrupts. At any time other than following these interupts, the programmer should get the most updated completion index values from the four completion registers.

TX MaxBurst#1, #2, #3, #4 (RW)

These 16bit registers are used to program the weights for the WRR arbitration of the four CBQ TX descriptor rings. The value in these registers represent the weights in bytes that would be transfered from the host memory to the TXFIFO in a round of the WRR arbitration. These registers can be updated dynamically. Upon writing a new value, the WRR is reset with the new weights upon completion of the current packet transfer from the host memory to the TXFIFO. Note that by a dummy write to any of these registers on the fly, is equivalent to queue1 pre-emption with all historical bandwidth defecit data reset to zero. This feature becomes handy incase the driver detects network congestion and requires a preemption or reallocation of network bandwidth.

TX FIFO Write Pointer (RW)

This 11-bit loadable counter points to the next location in the FIFO that will be loaded with TX data, the checksum or the frame control word. The counter is loaded with the contents of Shadow Write Pointer when the checksum is "stuffed" into the frame. The checksum offset within the 64-bit entry is accounted for when preparing the checksum to be stuffed. This counter is used to generate the "write" address for the TxFIFO memory core. Although this register is writeable, it should not be written into under normal operation. Writing is applicable in diagnostics.

TX FIFO Shadow Write Pointer (RW)

This 11-bit register points to the first byte of the packet that is either currently being loaded or is about to be loaded into the TX FIFO. This register serves as a temporary hold register for the Write Pointer and is used to "stuff" the checksum into the frame.Although this register is writeable, it should not be written into under normal operation. Writing is applicable in diagnostics.

TX FIFO Read Pointer (RW)

This 11-bit loadable counter points to the next location in the FIFO that will be read from to retrieve packet data that is transferred to the TX_MAC. The counter is loaded with the contents of the Shadow Read Pointer, when a "retry" occurs due to a collision on the network. This counter is used to generate the "read" address for the TX FIFO memory core.Although this register is writeable, it should not be written into under normal operation. Writing is applicable in diagnostics.

TX FIFO Shadow Read Pointer (RW)

This 11-bit register points to the first byte of the packet that is either currently being unloaded or is about to be unloaded from the TxFIFO. The register is loaded with the contents of the Read Pointer after the packet transfer from the FIFO to the TX_MAC has been completed. This register is used to "rewind" the Read Pointer for frame re-transmission due to a collision on the network.Although this register is writeable, it should not be written into under normal operation. Writing is applicable in diagnostics.

TX FIFO Packet Counter (RO)

This 11-bit up/down counter keeps track of the number of frames that currently reside in the TxFIFO. The counter increments when a frame is loaded into the FIFO, and decrements when a frame has been transferred to the TX_MAC. This counter is used to enable frame transfer from the TxFIFO to the TX_MAC.

TX State Machine Register#1 ,#2(RO)

This register provides the current state for all the state machines in TX.

Table 11-24 TX State Machine Register#1

Bits	Field Name	Description
[9:0]	Chain State	Describes the state of the Chaining state machine.
[11:10]	Checksum State	Describes the state of the Checksum state machine.
[17:12]	TX FIFO Load State	Describes the state of the TX FIFO Load State Machine. This field will have a value of 0x01 when the TX DMA is successfully disabled.
[21:18]	TX FIFO Unload State	Describes the state of the TX FIFO Unload State Machine.
[25:22]	Cache Controller State	Describes the state of the descriptor prefetch cache controller state machine.
[31:26]	CBQ Arbiter State	Describes the state of the CBQ Arbiter State machine.

Table 11-25 TX State Machine Register#2

Bits	Field Name	Description
[2:0]	Comp_Write_Back State	Describes the state of the completion Write Back State machine
[5:3]	Sub_load_State	Describes the state of the Sub load State machine.
[7:6]	Kick_State	Describes the state of the Kick State machine.

TX Data Pointer Low and High (RO)

A 64-bit pointer to the transmit data buffer in the host memory is stored in these two registers with the most significant bits in the High register. It is loaded by the DMA state machine, and it increments as the DMA reads in transmit data. Note that only the 50 least significant bits are incremented, while the upper 23 bits are taken as is from the TX descriptor.

TX FIFO Address (RW)

For diagnostic purposes a PIO path has been provided into the TX FIFO. This 11-bit register / counter holds the address of the 65 bit entry to be read / written.

TX FIFO Tag, Data Low, Data HighT0, and Data HighT1 (RW)

These registers are used for diagnostics access to any TX FIFO location. Every access involves 65 bits. The 32 least significant bits are accessed through the TX FIFO Data Low, the 32 most significant bits through either TX FIFO Data High T0 or T1 registers, while the TX FIFO Tag register is dedicated to reading the tag bit. Writing via the TX FIFO Data HighT0 results in the tag bit being written low, and writing via the TX FIFO Data HighT1 sets the tag bit high.

A write sequence consists of:

- Writing the FIFO address into TX FIFO Address Register
- Writing the TX FIFO Data Low and one of the TX FIFO Data High Registers. The write operation to the Data High Register triggers the 65 bit FIFO write.

A read sequence consists of:

- Writing the FIFO address into TX FIFO Address Register
- Reading from any of the TX FIFO Tag, Data Low, and Data High Registers. The read operation does not auto-increment the Address.

Programming Restrictions: *The TX_FIFO_PIO_Sel bit must be set to one for TX FIFO PIOs accesses. Prior to this, the TX DMA should be disabled by setting the TX_DMA_Enable bit to zero if TX FIFO data integrity is desired after the PIOs. The Data High should be the last access of the sequence.*

Programming Restrictions:

TX FIFO Size (RO)

This 11-bit read only register indicates the size, in 64 byte multiples, of the TX FIFO. The value of this register is 0x090, indicating a 9kbyte TX FIFO.

TX RAMBIST Control/Status Register (RW)

This 9-bit register is used to control the BIST operation of the TXFIFO memories. Bit[0] when set to one starts the BIST sequence , and upon completion of the testing, Bit[0] is self cleared by the ASIC. Clearing of Bit[0] is the indication to the simulation or diagnostic programs that the RAMBIST sequence is complete.

Table 11-26 Tx RAMBIST Control/Status Register

Bits	Field Name	Description
[8:6]	RAMBIST Controller State	Describes the Progress/State of the RAMBIST Controller State machine.
[5:5]	txflg33a	Indicates BIST pass/fail for RAM33A logic '1' indicates pass, else fail.
[4:4]	txflg32a	Indicates BIST pass/fail for RAM32A logic '1' indicates pass, else fail.
[3:3]	txflg33b	Indicates BIST pass/fail for RAM33B logic '1' indicates pass, else fail.

Table 11-26 Tx RAMBIST Control/Status Register

Bits	Field Name	Description
[2:2]	txflg32b	Indicates BIST pass/fail for RAM32B logic '1' indicates pass, else fail.
[1:0]	txflgsummary	Indicates BIST pass/fail for all RAMS logic '1' indicates pass, else fail.
[0:0]]	bist	Writing '1' into this field initiates the RAMBIST process, and is self cleared upon completion of the BIST process.

11.1.6 RX DMA Programmable Resources

RX Configuration Register (RW)

This register stores parameters that control the receive DMA channel operation.

*Table 11-27*RX Configuration Register

Bits	Field Name	Description
[0]	Rx_DMA_Enable	When set to '1', the RX DMA channel is enabled. When cleared to '0', the RX DMA channel will orderly stop operation as soon as the current frame transfer has been completed. As a practice the driver should disable the MAC for 200 ms before disabling the Rx.
[4:1]	Rx_Desc_Ring_Size	This field determines the number of descriptor entries in the Rx Free descriptor ring. Default: 0x8; 8k descriptor entries.
[8:5]	Rx_Comp_Ring_Size	This field determines the number of descriptor entries in the Rx Completion ring. Default: 0x8; 32k descriptor entries.
[9]	Batch_disable	When set to '1', disables receive descriptor "batching". Defaults low to batching enabled.
[12:10]	swivel_offset	This field determines the byte offset of the first data byte of the packet within 8 byte boundaries. This will swivel the data DMA'ed to header buffers, jumbo buffers when header split is not requested and MTU sized buffers. Defaults to 0x2.
[15:13]	Reserved	
[19:16]	Rx_Desc_Ring_Size_2	This field determines the number of descriptor entries in the Rx Free descriptor ring_2. Default: 0x8; 8k descriptor entries.
[31:20]	Reserved	

*Table 11-28*RX Descriptor Ring Size values

Rx_Desc_Ring_Size	Free Descriptor Ring Size	Completion Ring Size
0	32	128
1	64	256
2	128	512
3	256	1k
4	512	2k
5	1k	4k
6	2k	8k
7	4k	16k
8	8k	32k
9-15	Reserved	Reserved

Default: 0x80910

RX Parser Configuration Register (RW)

This register stores parameters that control the receive Header Parser operation.

*Table 11-29*RX Parser Configuration Register

Bits	Field Name	Description
[0]	Parsing Enabled	When set to '1', enable header parsing. Defaults to 0 - all packets will receive no Layer 3 hardware assist.
[1]	reserved	
[7:2]	Number of Processors	This field determines the mod value used to calculate the load balancing CPU number. Defaults to 0x1.
[8]	SYN Inc Mask	When set to '1', the presence of the SYN bit will not increment the TCP sequence number by one when stored in the FDBM. Defaults to 0.
[19:9]	TCP Payload Threshold	This field determines how many bytes of TCP data a packet needs to have to be considered for reassembly. Defaults to 1428 or 0x594.
[23:20]	reserved	

Default : 0x1651004.

The maximum number of processors the Cassini+ can support is 64. This is done by setting all bits in number of processors field to 0. Setting it to 1 means one processor, 2 means two processors, and 0x3f means 63 processors.

RX Page Size Register (RW)

[1:0]	Page Size	This field determines the size of the pages pointed to by the receive descriptors. If jumbo buffers is supported the page size should not be programmed to a value less than 8k. 2'b00 : 2k 2'b01 : 4k 2'b10 : 8k 2'b11 : 16k Default is 8k.
[10:2]	Reserved	
[14:11]	MTU Buffer Count	Driver writes into this field the number of MTU buffers the hardware packs within a page. Defaults to 4.
[21:15]	Reserved	
[28:27]	MTU Buffer Stride	The beginning of each MTU buffer prepended by N bytes encoded in the MTU offset byte field is separated from the beginning of the next MTU buffer with its offset bytes by value in this field. 2'b00 : 1k 2'b01 : 2k 2'b10 : 4k 2'b11 : 8k Defaults to 0x1
[29]	Reserved	
[31:30]	MTU Buffer Offset	Hardware writes an MTU buffer into a page offset by the number of bytes encoded in this field. Each buffer packed into a page has this offset prepended. Defaults to 0x1. 2'b00 : zero offset 2'b01 : 64 bytes 2'b10 : 96 bytes 2'b11 : 128 bytes

Default: 0x48002002

Note – Normally, the MTU Buffer Count is a function of the page size (e.g. page_size / MTU buffer Stride). However,it is not required that the page be fully utilized. If the MTU Buffer Count is programmed to a value which is too large than what would fit into the page (i.e. page_size / MTU_Buffer_Stride - MTU_Buffer_Count < 0) then the DMA will write over the page boundary. This would be a programming error.

RX FIFO Write Pointer (RW)

This 11-bit counter points to the next location in the RxFIFO that will be loaded with data from the RX IPP. The counter increments on 64-bit loads into the FIFO. This counter generates the "write" address for the RxFIFO memory core. Defaults to zero.

RX FIFO Read Pointer (RW)

This 11-bit counter points to the next location in the RxFIFO that will be read from to retrieve packet data to be transferred to host memory. This counter generates the "read" address for the RxFIFO memory core, on 64-bit boundaries. Defaults to zero.

IPP FIFO Write Pointer (RW)

This 11-bit counter points to the next location in the IPP FIFO that will be loaded with data from the RX_MAC. The counter increments on 64-bit loads into the FIFO. The counter is loaded with the contents of Shadow Write Pointer, when an "early receive abort" needs to be performed. This counter generates the "write" address for the IPP FIFO memory core. Defaults to zero.

IPP FIFO Shadow Write Pointer (RO)

This 11-bit register points to the first word of the packet that is currently being loaded into the FIFO. The register is loaded with the contents of the IPP Write pointer at the beginning of the packet transfer from the RX_MAC to the FIFO. This register is used to perform "early receive abort". Defaults to undefined.

IPP FIFO Read Pointer (RW)

This 11-bit counter points to the next location in the IPP FIFO that will be read from to retrieve packet data to be transferred to host memory. This counter generates the "read" address for the IPP FIFO memory core, on 64-bit boundaries. Defaults to 0x0.

Control/Batching FIFO Write Pointer (RW)

This 8-bit counter points to the next location in the Control FIFO that will be loaded with data from the RX IPP. The counter increments on 64-bit loads into the FIFO. This counter generates the "write" address for the Control FIFO memory core. Defaults to zero.

Control/Batching FIFO Read Pointer (RW)

This 8-bit counter points to the next location in the Control FIFO that will be read from to retrieve packet data to be transferred to host memory. This counter generates the "read" address for the Control FIFO memory core, on 64-bit boundaries. Defaults to zero.

RX Debug Register (RO)

This register provides the current state for the RX DMA state machines and other debug information.

Default: 0x00000000.

PAUSE Thresholds (RW)

Two PAUSE thresholds are used to define when PAUSE flow control frames are emitted by Cassini+ . XOFF PAUSE frames use the pause_time value pre-programmed in the Send PAUSE MAC Register, while XON PAUSE frames use a pause_time of zero. The granularity of these thresholds is in 64 byte increments. PAUSE thresholds are defined in terms of FIFO occupancy, and

Table 11-30 PAUSE Thresholds

Bits	Field Name	Description
[8:0]	OFF threshold	XOFF PAUSE emitted when RX FIFO occupancy rises above this value (times 64 bytes).
[11:9]	Reserved	
[20:12]	ON threshold	XON PAUSE emitted when after emitting XOFF PAUSE, the RX FIFO occupancy falls below this value (times 64 bytes). Must be smaller than the OFF threshold value. If equal to the RX FIFO Size, XON frames are never emitted.

may be translated into FIFO vacancy by the software driver using the RX FIFO Size register information. Programming the ON threshold to '1' will trigger the XON frames when the fifo level falls to zero. Since the threshold must be crossed to create the trigger, programming the value '0' to the ON threshold will never generate the trigger to the MAC to send XON frames. The OFF threshold should not be programmed above the size of the Rx Fifo. The maximum programmable value which will generate an XOFF frame is x6F.

Default: 0x00078.

RX Kick Register (RW)

This 13-bit register written by the host CPU. The last valid receive descriptor is the one right before the value of the register. The *RX Kick Register* and *RX Completion Register* values are used by the receive DMA engine to control RX descriptor fetching and interrupt generation. If the value indicates that four or more valid RX descriptors are available, Cassini+ will read four at time as needed, and internally cache them. Initialized to zero on reset.

Programming Restrictions: *Writing the value N in this register means that all descriptors up to but excluding N are available. Receive Descriptors must be posted in increments of four (N must be a multiple of four), and for best performance the first descriptor should be cache line aligned.*

RX Descriptor Ring Base High and Low (RW)

These registers form a 8KB aligned 64 bit pointer to the base of the RX free buffer descriptor ring. The lower 13 bits of the Low register are hardwired zero.

RX Completion Ring Base High and Low (RW)

These registers form a 8KB aligned 64 bit pointer to the base of the RX completion ring. The lower 13 bits of the Low register are hardwired zero.

RX Completion Register (RO)

This 13-bit register indicates which descriptors were already used by Cassini+ for receive frames. It is associated with the free descriptor ring. This register can be used in conjunction with the kick register to determine how many descriptors hardware has yet to use or it can be used as a diagnostic. Initialized to zero on reset.

RX Completion Head Register (RO)

This 13-bit register indicates which descriptors were already used by Cassini+ for receive frames. All descriptors between the completion head (including the descriptor pointed to by the head) and the completion tail (not including the descriptor pointed to by the tail) are owned by software. The DMA state machine increments the value of this register after the corresponding frame is entirely transferred from the RX FIFO into the host queue, but before setting the RX_DONE bit in the Interrupt Status Register. This register is used for diagnostic purposes. The driver uses the ownership bit in practice to determine the number of descriptors posted by hardware. Initialized to zero on reset.

RX Completion Tail Register (RW)

This 13-bit register written by the host CPU. The *RX Completion Head Register* and *RX Completion Tail Register* values are used by the receive DMA engine to control RX descriptor posting and interrupt generation. Immediately after reset, the completion tail is not to be programmed to a non-zero value. This will ensure that the entire completion ring can be available to the DMA for posting of descriptors. When the completion head has moved to indicate software owned entries exist in the completion ring (or sotware ownership bits in the completion descriptor indicate), the software can process the completion descriptors. After sotware has processed those completion descriptors, it can pass ownership back to hardware by incrementing the completion tail register to any point owned by software (see Completion Head Register). One may read the Completion Head to find out what is currently owned by hardware. To avoid this read for performance reasons, one can assume the completion head and tail are within 16 entries of each other when the completion full interrupt is given. The completion tail register should always trail the completion head register. To give all entries to hardware owndership the completion tail can be programmed to equal the completion head. If the completion head and tail values indicate that no more entries in the completion ring are available, the DMA will pause until room becomes available. An interrupt will be generated to indicate the lack of completion entries. Software can use this interrupt to reduce the number of times it must update the completion tail register. Initialized to zero on reset.

RX Blanking Register ISR (RW)

This register defines the values used for receive interrupt blanking, loaded each time the interrupt status register (ISR) is read.

Table 11-31 RX Blanking Register for ISR Read

Bits	Field Name	Description
[8:0]	RX_INTR_PACKETS	RX_DONE interrupt will be asserted if that many sets of completion writebacks (up to two packets will be represented per completion writeback) occur since the last time the interrupt status register was read. A value of zero indicates no packet blanking.
[11:9]	Reserved	
[29:12]	RX_INTR_TIME	RX_DONE interrupt will be asserted if that many clocks were counted since the last time the interrupt status register was read. Every count is 512 Core clocks (125 MHz). A value of zero indicates no time blanking.

Default: 0x00000.

RX Almost Empty Threshold Registers (RW)

This register defines the values used for interrupt generation based on threshold values of how many free descriptors and completion entries are available for hardware use.

Table 11-32 RX Almost Empty Threshold Register

Bits	Field Name	Description
[12:0]	Free Descriptors Low Threshold	Rx Buf Almost Empty interrupt will be asserted if the number of descriptors available for hardware use dwindles to this number.
[27:13]	Completion Entries Low Threshold	Rx Completion Almost Empty interrupt will be asserted if the number of completion entries available for hardware use dwindles to this number.
[31:28]	Reserved	

Default: 0x00000.

RX Random Early Detection Register (RW)

This register provides enables for each Random Early Drop threshold. When the fifo reaches a fullness falling into one of the RED ranges, packets are

Table 11-33 RX Random Early Detection Register

Bits	Field Name	Description
[7:0]	4k-6k RED pkt drop vector	Random Early Detection and Packet drop packet drop vectors for when fifo threshold is greater than 4096 bytes and less than 6,144 bytes. Probability of drop can be programmed on a 12.5% granularity. For example, it bit 0 is set the 1st packet out of every 8 will be dropped in this region.
[15:8]	6k-8k RED pkt drop vector	Random Early Detection and Packet drop packet drop vectors for when fifo threshold is greater than 6,144 bytes and less than 8,192 bytes. Probability of drop can be programmed on a 12.5% granularity. For example, it bit 8 is set the 1st packet out of every 8 will be dropped in this region.
[23:16]	8k-10k RED pkt drop vector	Random Early Detection and Packet drop enable for when fifo threshold is greater than 8,192 bytes and less than 10,240 bytes. Probability of drop can be programmed on a 12.5% granularity. For example, it bit 16 is set the 1st packet out of every 8 will be dropped in this region.
[31:24]	10k-12k RED pkt drop vector	Random Early Detection and Packet drop enable for when fifo threshold is greater than 10,240 bytes and less than 12,288 bytes. Probability of drop can be programmed on a 12.5% granularity. For example, it bit 24 is set the 1st packet out of every 8 will be dropped in this region.

dropped according to the preset probability. The probability should increase when the fifo level increases. Control packets are never dropped and are not counted in the statistics.

Default: 0x00000.

RX FIFO Fullness Register (RO)

This register provides fifo fullness levels for the Rx Fifo, Rx Control Fifo, and the Rx IPP fifo. The value of the Rx Control Fifo is equal to the number of packets in the Rx Fifo.

Rx Fifo level in 8B granularity	IPP fifo level in 8B granularity	Number of packets in Rx Fifo
[29:19]	[18:8]	bits [7:0]

Defaults to 0.

RX BIST Register (RO)

This register provides access for BIST testing for the Rx Fifo, Rx Control Fifo, and the Rx IPP fifo. Bits 1 and 0 can be used by software to trigger BIST and capture the result of the test of all 16 RAMs. Only the Rx BIST start/complete bit is writable.

Rx BIST 32a pass	Rx BIST 33a pass	Rx BIST 32b pass	Rx BIST 33b pass	Rx BIST 32c pass	Rx BIST 33c pass	Rx IPP BIST 32a pass	Rx IPP BIST 33a pass
bit[31]	bit [30]	bit[29]	bit[28]	bit[27]	bit[26]	bit[25]	bit[24]

Rx IPP BIST 32b pass	Rx IPP BIST 33b pass	Rx IPP IST 32c pass	Rx IPP BIST 33c pass	Rx Ctrl BIST 32 pass	Rx Ctrl BIST 33 pass	Rx Reas BIST 26a pass	Rx Reas BIST 26b pass
bit[23]	bit [22]	bit[21]	bit[20]	bit[19]	bit[18]	bit[17]	bit[16]

Rx Reas BIST 27 pass	Rx BIST state	reserved	Rx Bist Summary Pass	Rx BIST start/complete
bit[15]	bit[14:11]	bits [10:2]	bit [1]	[0]

Rx BIST 33a pass, Rx BIST 33a pass ... Rx Ctrl BIST 33 pass - the bits are results of each of the RAMs BIST test. They are valid when the BIST start/complete bit is cleared.

Rx BIST state - contain the state machine bits to debug any hang or malfunction in the BIST controller.

Rx Bist Summary Pass - when the BIST is complete, the BIST Summary Pass bit is valid. It contains the AND function of the individual BIST results of all 16 RAMs.

Rx BIST start/complete - when written to 1 by software, the BIST controller will begin testing Rx RAMs through BIST. The bit will remain high while BIST is being performed. The controller will proceed through the 17 RAMs checking BIST for pass and intentional fail. When the controller has completed checking the RAMs, it will set the bit to 0. It takes about 1,400 us to complete Rx BIST testing.

Defaults to 32'bxxxx_xxxx_xxxx_xxxx_xxxx_x000_0000_0000_00x0.

RX Control FIFO Write Pointer (RO)

This 8-bit counter points to the next location in the RxControl FIFO that will be loaded with data from the RX IPP. The counter increments per packet loaded into the RxFIFO. This counter generates the "write" address for the Rx Control FIFO. Defaults to zero.

RX Control FIFO Read Pointer (RO)

This 8-bit counter points to the next location in the Rx Control FIFO that will be read from to retrieve packet control information. This counter generates the "read" address for the Rx Control FIFO. Defaults to zero.

RX Blanking Register Alias (RW)

This register defines the values used for receive interrupt blanking, loaded each time the interrupt alias register is read.

Table 11-34 RX Blanking Register for Alias Read

Bits	Field Name	Description
[8:0]	RX_INTR_PACKETS	RX_DONE interrupt will be asserted if that many sets of completion writebacks (up to two packets will be represented per completion writeback) occur since the last time the interrupt status register was read. A value of zero indicates no packet blanking.
[11:9]	Reserved	
[29:12]	RX_INTR_TIME	RX_DONE interrupt will be asserted if that many clocks were counted since the last time the interrupt status register was read. Every count is 512 Core clocks (125 MHz). A value of zero indicates no time blanking.

Default: 0x00000.

RX FIFO Address (RW)

For diagnostic purposes a PIO path has been provided into the RX FIFO. This 11-bit register/counter holds the address of the 65 bit entry to be read/written. The value of this register does not auto-increment on write accesses to either RX FIFO Data High register. Defaults to undefined.

RX FIFO Tag, Data Low, Data HighT0, and Data HighT1 (RW)

These registers are used for diagnostics access to any of its 1536 Rx FIFO locations. Every access involves 65 bits. The 32 least significant bits are accessed through the RX FIFO Data Low, the 32 most significant bits through either RX FIFO Data High T0 or T1 registers, while the RX FIFO Tag register is dedicated to reading the tag bit. Writing via the RX FIFO Data HighT0 results in the tag bit being written low, and writing via the RX FIFO Data HighT1 sets the tag bit high. After diagnostics, in order to set the pointers to the correct location for normal operation, a write to address location 0 must occur.

A write sequence consists of:

- Writing the FIFO address into RX FIFO Address Register
- Writing the RX FIFO Data Low and one of the RX FIFO Data High Registers. The write operation to the Data High Register triggers the 65 bit FIFO write.

A read sequence consists of:

- Writing the FIFO address into RX FIFO Address Register
- Reading from any of the RX FIFO Tag, Data Low, and Data High Registers. The read operation does not auto-increment the Address.

Defaults to undefined.

Programming Restrictions: *The RX_DMA_Enable bit must be set to zero for RX FIFO PIOs accesses.The Data High should be the last write access of the write sequence.*

RX Control Fifo and Batching Table Address (RW)

For diagnostic purposes a PIO path has been provided into the RX Control Fifo. This 8-bit register/counter holds the address of the 81 bit control entry and the 6 bit flow_id to be read/written. Defaults to undefined.

RX Control Fifo Data Low, Mid, High (RW)

These registers are used for diagnostics access to any RX Control Fifo location. Every access involves 32 bits except for access to the Data High Register during which the bits[6:0] are accessed. After diagnostics, in order to set the pointers to the correct location for normal operation, a write to address location 0 must occur. A write sequence consists of:

- Writing the Rx Control Fifo Register with a value of 0 to 255.
- Writing the RX Control Fifo Data Low Register.
- Writing the RX Control Fifo Data Mid Register.
- Writing the RX Control Fifo Data High Register [6:0] where bit [0] contains the upper bit of the control word and bits[6:1] contain the flow_id.

A read sequence consists of:

- Writing the Rx Control Fifo Register with a value of 0 to 255.
- Reading from the the RX Control Fifo Data Low Register.
- Reading from the the RX Control Fifo Data Mid Register.
- Reading from the the RX Control Fifo Data High Register bits [6:0].

Defaults to undefined.

Programming Restrictions: *The RX_DMA_Enable bit must be set to zero for* Rx Control Fifo Register *PIOs accesses.The Data High should be the last write access of the write sequence.*

RX IPP FIFO Address (RW)

For diagnostic purposes a PIO path has been provided into the RX FIFO. This 9-bit register/counter holds the address of the 65 bit entry to be read/written. Defaults to undefined.

RX IPP FIFO Tag, Data Low, Data HighT0, and Data HighT1 (RW)

These registers are used for diagnostics access to any of its 1280 RX IPP FIFO locations. Every access involves 65 bits. The 32 least significant bits are accessed through the RX IPP FIFO Data Low, the 32 most significant bits through either RX IPP FIFO Data High T0 or T1 registers, while the RX IPP FIFO Tag register is dedicated to reading the tag bit. Writing via the RX IPP FIFO Data HighT0 results in the tag bit being written low, and writing via the RX IPP FIFO Data HighT1 sets the tag bit high. After diagnostics, in order to set the pointers to the correct location for normal operation, a write to address location 0 must occur.

A write sequence consists of:

- Writing the FIFO address into RX IPP FIFO Address Register
- Writing the RX IPP FIFO Data Low Register.
- Writing the RX IPP FIFO Data High Register. The write operation to the Data High Register triggers the 65 bit FIFO write.

A read sequence consists of:

- Writing the FIFO address into RX IPP FIFO Address Register
- Reading from any of the RX IPP FIFO Tag, Data Low, and Data High Registers.

Defaults to undefined.

The RX_DMA_Enable bit must be set to zero for RX IPP FIFO PIOs accesses.The Data High should be the last write access of the write sequence.

RX Header Page Address High and Low Registers (RO)

A 64-bit pointer to the receive data buffer in the host memory used for headers and small packets is stored in these two registers with the most significant bits in the High register. It is loaded by the DMA state machine, and it increments as the DMA writes receive data. Note that only the 50 least significant bits are incremented, while the upper 13 bits are taken as is from the RX descriptor. Defaults to undefined.

RX MTU Page Address High and Low Registers (RO)

A 64-bit pointer to the receive data buffer in the host memory used for headers and small packets is stored in these two registers with the most significant bits in the High register. It is loaded by the DMA state machine, and it increments as the DMA writes receive data. Note that only the 50 least significant bits are incremented, while the upper 13 bits are taken as is from the RX descriptor. Defaults to undefined.

RX Reassembly DMA Table RAM Addresses (RW)

For diagnostic purposes a PIO path has been provided into the RX Reassembly DMA Table RAM. This 6-bit register holds the address of one of the 64 79-bit locations in the RX Reassembly DMA Table and the address of one of the sixty-four byte location in the Batching Table. The data fills the least significant

Bits	Field Name	Description
[5:0]	RX Reassembly Table Address	Selects one of sixty-four 79-bit locations in the RX Reassembly Table to read or write.
[31:6]	Reserved	

portion of the bus if it is less than 32 bits. Defaults to undefined.

RX Reassembly DMA Table Data Low, Data Mid, Data High (RW)

RX Reassembly DMA Table location. Every write access involves 79 bits. The 32 least significant bits are accessed through the RX Reassembly DMA Table Data Low, the 32 next least significant bits are accessed through the RX Reassembly DMA Table Data Mid, the 15 most significant bits through either RX Reassembly DMA Table Data High register. A write sequence consists of:

- Writing the RX Reassembly DMA Table Address Register bits[5:0] with a value of 0 to 64.
- Writing the RX Reassembly DMA Table Data Low Register
- Writing the RX Reassembly DMA Table Data Mid Register
- Writing the RX Reassembly DMA Table Data High Register bits[14:0].

A read sequence consists of:

- Writing the RX Reassembly DMA Table Address Register bits[5:0] with a value of 0 to 64.
- Reading from the RX Reassembly DMA Table Data Low
- Reading from the RX Reassembly DMA Table Data Mid Register
- Reading from the RX Reassembly DMA Table Data High Register bits[14:0].

The RX_DMA_Enable bit must be set to zero for RX Reassembly DMA Table PIOs accesses.The Data High should be the last write access of the write sequence.

reassembly ptr	reassembly index	reassembly entry valid
bits[78:15]	bits[14:1]	bit[0]

Following 17 Rx registers are for Cassini+ .

Programming Restrictions: *When using one or both descriptor rings, the registers of both descriptor rings must be initialized. Failure to do this will result in inproper Cassini+ rx dma state machine operation.*

RX Descriptor Ring 2 Base High and Low (RW)

These registers form a 8KB aligned 64 bit pointer to the base of the RX free buffer descriptor ring. The lower 13 bits of the Low register are hardwired zero. This is for rx_free_descriptor ring_2.

RX Completion Ring Base 2 High and Low (RW)

These registers form a 8KB aligned 64 bit pointer to the base of the RX completion ring. The lower 13 bits of the Low register are hardwired zero. This is for rx_completion ring_2.

RX Completion Ring Base 3 High and Low (RW)

These registers form a 8KB aligned 64 bit pointer to the base of the RX completion ring. The lower 13 bits of the Low register are hardwired zero. This is for rx_completion ring_3.

RX Completion Ring Base 4 High and Low (RW)

These registers form a 8KB aligned 64 bit pointer to the base of the RX completion ring. The lower 13 bits of the Low register are hardwired zero. This is for rx_completion ring_4.

RX Kick 2 Register (RW)

This 13-bit register written by the host CPU. The last valid receive descriptor is the one right before the value of the register. The *RX Kick Register* and *RX Completion Register* values are used by the receive DMA engine to control RX descriptor fetching and interrupt generation. If the value indicates that four or more valid RX descriptors are available, Cassini+ will read four at time as needed, and internally cache them. Initialized to zero on reset. This is for rx_free_descriptor_ring 2.

Programming Restrictions: *Writing the value N in this register means that all descriptors up to but excluding N are available. Receive Descriptors must be posted in increments of four (N must be a multiple of four), and for best performance the first descriptor should be cache line aligned.*

RX Completion 2 Register (RO)

This 13-bit register indicates which descriptors were already used by Cassini+ for receive frames. It is associated with the free descriptor ring. This register can be used in conjunction with the kick register to determine how many descriptors hardware has yet to use or it can be used as a diagnostic. Initialized to zero on reset. This is for rx_free_descriptor_ring 2.

RX Completion Head 2 Register (RO)

This 13-bit register indicates which descriptors were already used by Cassini+ for receive frames. All descriptors between the completion head (including the descriptor pointed to by the head) and the completion tail (not including the descriptor pointed to by the tail) are owned by software. The DMA state machine increments the value of this register after the corresponding frame is entirely transferred from the RX FIFO into the host queue, but before setting the RX_DONE bit in the Interrupt Status Register. This register is used for diagnostic purposes. The driver uses the ownership bit in practice to determine the number of descriptors posted by hardware. Initialized to zero on reset. This is for rx_completion_ring_2.

RX Completion Tail 2 Register (RW)

This 13-bit register written by the host CPU. The *RX Completion Head Register* and *RX Completion Tail Register* values are used by the receive DMA engine to control RX descriptor posting and interrupt generation. Immediately after reset, the completion tail is not to be programmed to a non-zero value. This will ensure that the entire completion ring can be available to the DMA for posting of descriptors. When the completion head has moved to indicate software owned entries exist in the completion ring (or sotware ownership bits in the completion descriptor indicate), the software can process the completion descriptors. After sotware has processed those completion descriptors, it can pass ownership back to hardware by incrementing the completion tail register to any point owned by software (see Completion Head Register). One may read the Completion Head to find out what is currently owned by hardware. To avoid this read for performance reasons, one can assume the completion head and tail are within 16 entries of each other when the completion full interrupt is given. The completion tail register should always trail the completion head register. To give all entries to hardware owndership the completion tail can be programmed to equal the completion head. If the completion head and tail values indicate that no more entries in the completion ring are available, the DMA will pause until room becomes available. An interrupt will be generated to indicate the lack of completion entries. Software can use this interrupt to reduce the number of times it must update the completion tail register. Initialized to zero on reset. This is for rx_completion_ring_2.

RX Completion Head 3 Register (RO)

This 13-bit register indicates which descriptors were already used by Cassini+ for receive frames. All descriptors between the completion head (including the descriptor pointed to by the head) and the completion tail (not including the descriptor pointed to by the tail) are owned by software. The DMA state machine increments the value of this register after the corresponding frame is entirely transferred from the RX FIFO into the host queue, but before setting the RX_DONE bit in the Interrupt Status Register. This register is used for diagnostic purposes. The driver uses the ownership bit in practice to determine the number of descriptors posted by hardware. Initialized to zero on reset. This is for rx_completion_ring_3.

RX Completion Tail 3 Register (RW)

This 13-bit register written by the host CPU. The *RX Completion Head Register* and *RX Completion Tail Register* values are used by the receive DMA engine to control RX descriptor posting and interrupt generation. Immediately after reset, the completion tail is not to be programmed to a non-zero value. This will ensure that the entire completion ring can be available to the DMA for posting of descriptors. When the completion head has moved to indicate software owned entries exist in the completion ring (or sotware ownership bits in the completion descriptor indicate), the software can process the completion descriptors. After sotware has processed those completion descriptors, it can pass ownership back to hardware by incrementing the completion tail register to any point owned by software (see Completion Head Register). One may read the Completion Head to find out what is currently owned by hardware. To avoid this read for performance reasons, one can assume the completion head and tail are within 16 entries of each other when the completion full interrupt is given. The completion tail register should always trail the completion head register. To give all entries to hardware owndership the completion tail can be programmed to equal the completion head. If the completion head and tail values indicate that no more entries in the completion ring are available, the DMA will pause until room becomes available. An interrupt will be generated to indicate the lack of completion entries. Software can use this interrupt to reduce the number of times it must update the completion tail register. Initialized to zero on reset. This is for rx_completion_ring_3.

RX Completion Head 4 Register (RO)

This 13-bit register indicates which descriptors were already used by Cassini+ for receive frames. All descriptors between the completion head (including the descriptor pointed to by the head) and the completion tail (not including the descriptor pointed to by the tail) are owned by software. The DMA state machine increments the value of this register after the corresponding frame is entirely transferred from the RX FIFO into the host queue, but before setting the RX_DONE bit in the Interrupt Status Register. This register is used for diagnostic purposes. The driver uses the ownership bit in practice to determine the number of descriptors posted by hardware. Initialized to zero on reset. This is for rx_completion_ring_4.

RX Completion Tail 4 Register (RW)

This 13-bit register written by the host CPU. The *RX Completion Head Register* and *RX Completion Tail Register* values are used by the receive DMA engine to control RX descriptor posting and interrupt generation. Immediately after reset, the completion tail is not to be programmed to a non-zero value. This will ensure that the entire completion ring can be available to the DMA for posting of descriptors. When the completion head has moved to indicate software owned entries exist in the completion ring (or sotware ownership bits in the completion descriptor indicate), the software can process the completion descriptors. After sotware has processed those completion descriptors, it can pass ownership back to hardware by incrementing the completion tail register to any point owned by software (see Completion Head Register). One may read the Completion Head to find out what is currently

owned by hardware. To avoid this read for performance reasons, one can assume the completion head and tail are within 16 entries of each other when the completion full interrupt is given. The completion tail register should always trail the completion head register. To give all entries to hardware owndership the completion tail can be programmed to equal the completion head. If the completion head and tail values indicate that no more entries in the completion ring are available, the DMA will pause until room becomes available. An interrupt will be generated to indicate the lack of completion entries. Software can use this interrupt to reduce the number of times it must update the completion tail register. Initialized to zero on reset. This is for rx_completion_ring_4.

RX Almost Empty Threshold 2 Registers (RW)

This register defines the values used for interrupt generation based on threshold values of how many free descriptors and completion entries are available for hardware use. This is for rx_free_descriptor 2.

Table 11-35 RX Almost Empty Threshold Register

Bits	Field Name	Description
[12:0]	Free Descriptors Low Threshold	Rx Buf Almost Empty interrupt will be asserted if the number of descriptors available for hardware use dwindles to this number.
[31:13]	Reserved	

Default: 0x00000.

RX Instruction RAM Address (RW)

For diagnostic purposes a PIO path has been provided into the RX Instruction RAM. This 5-bit register/counter holds the address of the 39 bit entry to be read/written. Defaults to undefined.

RX Instruction RAM Data Low, Data High (RW)

These registers are used for diagnostics access to any RX Instruction RAM location. Every access involves 39 bits. The 32 least significant bits are accessed through the RX Instruction RAM Data Low, the 7 most significant bits through either RX Instruction RAM Data High register. A write sequence consists of:

- Writing the Rx Instruction RAM Register with a value of 0 to 31.
- Writing the RX FIFO Data Low Register.
- Writing the RX FIFO Data High Register.

A read sequence consists of:

- Writing into the RX Instruction RAM Address Register
- Reading from the RX Instruction RAM Data Low and Data High Registers.

Defaults to undefined.

Programming Restrictions: *The RX_DMA_Enable bit must be set to zero for RX* Instruction RAM *PIOs accesses.The Data High should be the last access of the sequence.*

RX Parser Data RAM and Flow Database Addresses (RW)

For diagnostic purposes a PIO path has been provided into the RX Header Parser Data RAM and Flow Database. This 11-bit register holds the address of one of the 31 4-byte locations in the Rx Header Parser Data RAM and the address of one of the sixty-four 353-bit location in the Flow Database. The data

Bits	Field Name	Description
[4:0]	Rx Parser Data RAM Address	Selects one of 86 byte locations in the Header Parser Data RAM to read or write.
[7]	Reserved	
[13:8]	Rx Flow Database Address	Selects one of sixty-four 353-bit locations

fills the least significant portion of the bus if it is less than 32 bits. Defaults to undefined.

RX Parser RAM Data (RW)

These registers are used for diagnostics access to any RX Parser Data RAM location. Every access involves 32 bits. A write sequence consists of:

- Writing the Rx Parser Data RAM Address Register bits[4:0] with a value of 0 to 31. An address of zero will give bytes 0-3 in the Data Ram. An addres of 31 will give bytes 120-123 in the Data RAM.
- Writing the RX Parser Data Register bits [31:0].

A read sequence consists of:

- Writing the Rx Parser Data RAM Address Register bits[4:0] with a value of 0 to 31.
- Reading from the RX Parser RAM Data Register bits [31:0].

Defaults to undefined.

Programming Restrictions: *The RX_DMA_Enable bit must be set to zero for RX* Parser RAM *PIOs accesses.The RX Parser RAM Data Register should be the last write access of the write sequence.*

RX Flow Database Data1..12 (RW)

These registers are used for diagnostics access to any RX Flow Database location. A write sequence consists of:

- Writing the Rx Flow Database Address Register bits[13:8] with a value of 0 to 63.
- Writing the RX Flow Database Register 1 with a value for IP_SA[127:96].

- Writing the RX Flow Database Register 2 with a value for IP_SA[95:64].
- Writing the RX Flow Database Register 3 with a value for IP_SA[63:32].
- Writing the RX Flow Database Register 4 with a value for IP_SA[31:0].
- Writing the RX Flow Database Register 5 with a value for IP_DA[127:96].
- Writing the RX Flow Database Register 6 with a value for IP_DA[95:64].
- Writing the RX Flow Database Register 7 with a value for IP_DA[63:32].
- Writing the RX Flow Database Register 8 with a value for IP_DA[31:0].
- Writing the RX Flow Database Register 9 with a value for {TCP_SP[15:0],TCP_DP[15:0]}.
- Writing the RX Flow Database Register 10 bit [0] with a value for Flow_Valid.
- Writing the RX Flow Database Register 11 with a value for TCP_SEQ[63:31].
- Writing the RX Flow Database Register 12 with a value for TCP_SEQ[31:0].

A read sequence consists of:

- Writing the Rx Flow Database Address Register bits[13:8] with a value of 0 to 63..
- Reading the RX Flow Database Register 1 with a value for IP_SA[127:96].
- Reading the RX Flow Database Register 2 with a value for IP_SA[95:64].
- Reading the RX Flow Database Register 3 with a value for IP_SA[63:32].
- Reading the RX Flow Database Register 4 with a value for IP_SA[31:0].
- Reading the RX Flow Database Register 5 with a value for IP_DA[127:96].
- Reading the RX Flow Database Register 6 with a value for IP_DA[95:64].
- Reading the RX Flow Database Register 7 with a value for IP_DA[63:32].
- Reading the RX Flow Database Register 8 with a value for IP_DA[31:0].
- Reading the RX Flow Database Register 9 with a value for {TCP_SP[15:0],TCP_DP[15:0]}.
- Reading the RX Flow Database Register 10 bit[0] with a value for Flow_Valid.
- Reading the RX Flow Database Register 11 with a value for TCP_SEQ[63:31].
- Reading the RX Flow Database Register 12 with a value for TCP_SEQ[31:0].

Defaults to undefined.

The RX_DMA_Enable bit must be set to zero for RX Flow Database *PIOs accesses.The RX Flow Database Data Register 12 should be the last write access of the write sequence.*

RX Parser State Machine Register (RO)

The Rx Header Parser block provides diagnostics through its state machine register.

RX Parser Status Register 1 (RO)

The Rx Parser Status register contains the values of its local registers.

SAP	L3 Offset	Load Balancing CPU Number	HRP Opcode
[31:16]	[15:9]	[8:3]	[2:0]

RX Parser Status Register 2 (RO)

The Rx Parser Status register contains the values of its local registers.

accu_R2[6:4]	flow id	TCP Payload Offset	TCP Payload Size
[31:29]	[28:23]	[22:16]	[15:0]

RX Parser Status Register 3 (RO)

The Rx Parser Status register contains the values of its local registers.

accu_R2 [3:0]	reserved	checksum start offset	accu_R1	force drop	batching w/o reassembly	jumbo header split enable	force TCP no payload check	mask of data length equal to zero
[31:28]	[27]	[26:20]	[19:13]	[12]	[11]	[10]	[9]	[8]

force TCP payload check	mask of payload threshhold	no_assist	control packet flag	tcp flag check	syn flag	tcp payload check	tcp no payload check
[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]

HP RAMBIST Control/Status Register (RW)

This register controls the BIST operation of the Header Parser (HP) and Flow Database Management (FDBM) memories. Setting bit[0] one starts the BIST sequence , and upon completion of the testing, the bist controller resets bit[0] to 0. Bit[0] can also be cleared by a diagnostic program to force the termination of the bist operation.

Bits	Field Name	Description
[31]	RAM 16x64 test pass	HP data RAM passes all bist tests.
[30]	RAM 32x32_0 test pass	HP instruction RAM0 passes all bist tests.
[29]	RAM 32x32_1 test pass	HP instruction RAM1 passes all bist tests.
[28]	RAM 32x32_2 test pass	HP instruction RAM2 passes all bist tests.
[27]	RAM 32x32_3 test pass	FDBM aging RAM0 passes all bist tests.
[26]	RAM 32x32_4 test pass	FDBM aging RAM1 passes all bist tests.
[25]	RAM 32x72_0 test pass	FDBM flowid RAM0 bank 0 passes all bist tests.
[24]	RAM 32x72_1 test pass	FDBM flowid RAM1 bank 0 passes all bist tests.

Bits	Field Name	Description
[23]	RAM 32x72_2 test pass	FDBM flowid RAM2 bank 0 passes all bist tests.
[22]	RAM 32x72_3 test pass	FDBM flowid RAM3 bank 0 passes all bist tests.
[21]	RAM 32x72_4 test pass	FDBM flowid RAM0 bank 1 passes all bist tests.
[20]	RAM 32x72_5 test pass	FDBM flowid RAM1 bank 1 passes all bist tests.
[19]	RAM 32x72_6 test pass	FDBM flowid RAM2 bank 1 passes all bist tests.
[18]	RAM 32x72_7 test pass	FDBM flowid RAM3 bank 1 passes all bist tests.
[17]	RAM 64x32 test pass	FDBM tcp sequence RAM passes all bist tests.
[16:2]	reserved	
[1]	Bist Pass Summary	Set to 1 when all bist tests pass.
[0]	Bist Start/Complete	Set to 1 to start HP Bist operation. It is self cleared upon completion.

11.1.7 MAC Programmable Resources

11.1.7.1 Command Registers

These registers are used by the software to instruct the hardware that a certain hardware function is to be executed upon the detection of the command bit. The command bits are set to '1' using a PIO write, and are "self-cleared" after the command execution has completed. A command bit can be polled at any time using a PIO read to verify the command execution.

TX_MAC Software Reset Command (RW)

This 1-bit command performs a software reset to the logic in the TX_MAC. The bit is set to '1' when a Programmed IO write is performed to the defined address. This bit becomes "self-cleared" after the command has been executed.

Default: 0x0.

RX_MAC Software Reset Command (RW)

This 1-bit command performs a software reset to the logic in the RX_MAC. The bit is set to '1' when a Programmed IO write is performed to the defined address. This bit becomes "self-cleared" after the command has been executed.

Default: 0x0.

Send Pause Command (RW)

This command register executes a Pause Flow Control frame transmission.

[15:0]:	Pause_Time_Sent:	This control field indicates to the MAC the value of the pause_time operand that should be sent on the network using either the Send_Pause command bit or the flow control handshake on the RxDMA<->MAC interface. The pause_time is interpreted in units of Slot Times.
[16]:	Send_Pause:	This command bit is used by the software to instruct the MAC that a Pause Flow Control frame should be sent on the network.

Default: 0x0XXXX.

11.1.7.2 Status and Mask Registers

These registers are used by the hardware to communicate to the software the occurrence of events that were detected by the hardware. A status register may contain interrupt bits and information fields.
If an interrupt bit is set to '1', it indicates the occurrence of the corresponding event. Interrupt bits are auto-cleared when the status register is read by the software, and have corresponding mask bits in a mask register. If a mask bit is cleared to '0', the corresponding status event will generate an interrupt. Mask Registers default to all ones upon reset.

Information fields are provided to complement interrupt status bits. They are unaffected by PIO reads and cannot be masked.
Status bits default to all 0's, and the corresponding mask bits default to all 1's.

TxMAC Status Register (R-AC)

[0]: Frame_Transmitted: This interrupt status bit indicates the successful transmission of a frame on the network.

[1]: Tx_Underrun: This interrupt status bit indicates that the MAC has terminated a frame transmission due to "data starvation" in the transmit data path.

[2]: Max_Pkt_Err: This interrupt status bit indicates that a frame that exceeds the maximum allowed length was passed to the TxMAC by the DMA engine.

[3]: Normal_Coll_Cnt_Exp: This interrupt status bit indicates the roll over of the Normal Collision Counter.

[4]: Excess_Coll_Cnt_Exp: This interrupt status bit indicates the roll over of the Excessive Collision Counter.

[5]: Late_Coll_Cnt_Exp: This interrupt status bit indicates the roll over of the Late Collision Counter.

[6]: First_Coll_Cnt_Exp: This interrupt status bit indicates the roll over of the First Collision Counter.

[7]: Defer_Timer_Exp: This interrupt status bit indicates the roll over of the Defer Timer.

[8]: Peak_Attmpt_Cnt_Exp: This interrupt status bit indicates the roll over of the Peak Attempts Counter.

RxMAC Status Register (R-AC)

[0]: Frame_Received: This interrupt status bit indicates the successful reception of a frame from the network.

[1]: Rx_Overflow: This interrupt status bit indicates that the MAC has dropped a receive frame due to the lack of resources in the receive data path (RX FIFO overflow).

[2]: Frame_Cnt_Exp: This interrupt status bit indicates the roll over of the Receive Frame Counter.

[3]: Align_Err_Cnt_Exp: This interrupt status bit indicates the roll over of the Alignment Error Counter.

[4]: CRC_Err_Cnt_Exp: This interrupt status bit indicates the roll over of the CRC Error Counter.

[5]: Length_Err_Cnt_Exp: This interrupt status bit indicates the roll over of the Length Error Counter.

[6]: Viol_Err_Cnt_Exp: This interrupt status bit indicates the roll over of the Code Violation Error Counter.

MAC Control Status Register (R-AC)

[0]: Pause_Received: This interrupt status bit indicates the successful reception of a Pause Flow Control frame from the network.

[1]: Paused_State: This interrupt status bit indicates that the MAC has made a state transition from "not-paused" to "paused".

[2]: Not-Paused_State: This interrupt status bit indicates that the MAC has made a state transition from "paused" to "not-paused".

[15:3]: Reserved.

[31:16]: Pause_Time_Rcvd: This information field indicates the value of the "pause_time" operand that was received in the last Pause Flow Control frame.

After reset, the upper 16 bits default to "x" and the lower 16 bits default to zero.

TxMAC Mask Register (RW)

The layout of this register corresponds bit-by-bit to the layout of the TxMAC Status Register, bits [8:0].

RxMAC Mask Register (RW)

The layout of this register corresponds bit-by-bit to the layout of the RxMAC Status Register, bits [6:0].

MAC Control Mask Register (RW)

The layout of this register corresponds bit-by-bit to the layout of the MAC Control Status Register, bits [2:0].

11.1.7.3 Configuration Registers

XIF Configuration Register (RW)

This register determines the parameters that control the operation of the transceiver interface.

Table 11-36 XIF Configuration Register

Bits	Field Name	Description
[0]	Tx_MII_OE	When set to '1', this bit enables the output drivers on the MII transmit bus
[1]	MII_Int_Loopback	When this bit is set to '1', the GMII transmit data path is internally (on-chip) looped back to the GMII receive data path. Consequently, the phy_mode register clock selection must be set to GMII mode and bit 3 of this XIF register must be set to 1 (GMII mode). Additionally, in loopback mode, the REFCLK will drive the entire mac core. This bit must be low for normal operation.
[2]	Disable_Echo	This bit should be enabled only in MII Giga Half Duplex mode or MII 10/100 Half Duplex mode (when an external PHY transceiver is used in half duplex mode). When set to 1, this bit disables the receive data path during packet transmission by the TxMAC. Effectively, the RX_DV signal from the MII is forced to 0, when the TX_EN is active. This bit should be cleared to 0 when the MAC operates in any Full Duplex mode, when it operates in any Loopback mode (Internal or External), or is in Half Duplex Serdes or Slink modes.
[3]	GMIIMODE	When set it configures the MAC interface to operate with GMII clocks and datapath. When clear it operates with MII clocks and datapath.
[4]	MII_Buffer_OE	This bit controls the MII_BUF_EN pin. This bit when set is intended to enable the external tri-state buffer that may reside on the MII receive data bus. Multiple transceivers may be connected to the MAC using this mechanism. When this bit is '0' the pin is low and the buffer is disabled.
[5]	LINKLED	When set it forces the LINKLED# active (low).
[6]	FDPLXLED	When set it forces the FDPLXLED# active (low).

Default: 0x0A.

Programming Restrictions: *To ensure proper operation of the hardware, when a loopback configuration is entered or exited, a Global Initialization Sequence should be performed. Additionally, programming of this register post hardware or global software reset, should be*

followed by a mac rx/tx software reset and initialization. This will ensure stable clocking prior to mac initialization, preventing clock glitches.

TX_MAC Configuration Register (RW)

This register controls the operation of the TX_MAC.

Table 11-37 TX_MAC Configuration Register

Bits	Field Name	Description
[0]	TxMAC_Enable	When set to '1', the TxMAC starts requesting packet data from the OPP, and the Ethernet transmit protocol execution begins. When cleared to '0', it will force the TxMAC state machines to either remain in the idle state, or to transition to the idle state and stay there at the completion of an ongoing packet transmission.
[1]	Ignore_Carrier_Sense	When set to '1', this bit causes the TxMAC to disable the CSMA/CD deferral process. This bit should be set to '1' while in Full Duplex mode, and cleared to '0' while in Half Duplex mode of operation.
[2]	Ignore_Collisions	When set to '1', this bit causes the TxMAC to disable the CSMA/CD backoff algorithm. This bit should be set to '1' while in Full Duplex mode, and cleared to '0' while in Half Duplex mode of operation.
[3]	Enable_IPG0	When set to '1', this bit enables the extension of the Rx-to-Tx IPG. After receiving a frame, the TxMAC will reset its deferral process in response to Carrier Sense assertion during the period of time that is the sum of the values programmed in the IPG0 and IPG1 registers, and will commit to transmission during the period of time that is programmed in the IPG2 register. When cleared to '0', or when transmitting frames back-to-back (Tx-to-Tx IPG), the TxMAC ignores IPG0. It will reset its deferral process in response to Carrier Sense assertion during the period of time that is programmed in the IPG1 register, and will commit to transmission during the period of time that is programmed in the IPG2 register.

Table 11-37 TX_MAC Configuration Register

Bits	Field Name	Description
[4]	Never_Give_Up	When set to '1', this bit modifies the CSMA/CD protocol, such that the TxMAC will not "easily give up" on a frame transmission. If the backoff algorithm reaches the attempts_limit, it will clear the attempts_counter and will continue trying to send the frame for an extended period of time (as defined by bit [5]). When this bit is cleared to '0', the TxMAC will execute the standard CSMA/CD protocol.
[5]	Never_Give_Up_Limit	When this bit is set to '1', the TxMAC continues trying to send out the frame until it is successfully transmitted on the medium. In effect, no limit will exist on the number of transmission attempts. When this bit is cleared to '0', the TxMAC will continue trying to transmit the frame until it is either successfully transmitted on the medium, or the backoff algorithm reached the attempts_limit for sixteen times.
[6]	No_Backoff	When set to '1', this bit modifies the CSMA/CD protocol, such that the backoff algorithm in the Protocol Engine is disabled. The TxMAC will not back off after a transmission attempt that resulted in a collision on the medium. Effectively, the random number, generated by the backoff algorithm, is always fixed to '0'. When this bit is cleared to '0', the TxMAC will execute the standard CSMA/CD protocol.
[7]	Slow_Down	When set to '1', this bit modifies the CSMA/CD protocol, such that the TxMAC resets its deferral process in response to the assertion of the Carrier Sense during the entire duration of the IPG. In effect, the TxMAC commits to the transmission of a frame only after the frame transmission has actually begun. When this bit is cleared to '0', the TxMAC will execute the standard CSMA/CD protocol.
[8]	No_FCS	When set to '1', the TxMAC will not generate the CRC for all transmitted packets. When clear to '0' the TxMAC will act on the CRC generation as indicated by the NO_CRC bit in the transmit Control Word from the TxDMA.
[9]	TX_Carrier_Extension	When set to '1', this bit enables the transmit part of the Carrier Extension feature. This feature allows longer collision domains by extending if necessary the carrier and the collision window from the end of the FCS until the end of the Slot Time. Carrier Extension is required for half-duplex operation at 1 Gbps, this bit should be clear otherwise.

Programming Restrictions: *To ensure proper operation of the TX_MAC, the TX_MAC_Enable bit must always be cleared to '0' and a delay imposed before a PIO write to any of the other bits in the TX_MAC Configuration register or any of the MAC parameters registers is performed.*

The amount of delay required will depend on the time required to transmit a maximum size frame, and is thus dependent on the value programmed into the MaxFrameSize register and the data rate on the medium. For a standard 1518-byte frame on a 100Mbps network the delay would be 125usec. To avoid the requirement for a variable time delay, the TX_MAC_Enable bit may be polled, and when this bit reads back as a '0', all the registers mentioned above may be written, including all the other bits in the Configuration register.

Note – Enabling Carrier Extension involves setting the TX_Carrier_Extension, RX_Carrier_Extension bits and configuring the Slot Time Register to 0x200 (Slot Time of 512 bytes). This mode must be enabled whenever Cassini operates in Half-Duplex at 1Gbps, and disabled otherwise.

RX_MAC Configuration Register (RW)

This register controls the operation of the RX_MAC.

Table 11-38 RX_MAC Configuration Register

Bits	Field Name	Description
[0]	Rx_MAC_Enable	When set to '1', the RX_MAC is enabled. When cleared to '0', it will force the RX_MAC state machines to either remain in the idle state, or to transition to the idle state and stay there
[1]	Strip_Pad	Always program Strip_Pad bit to zero. This feature is no longer supported.
[2]	Strip_FCS	When set to '1', this bit will cause the RxMAC to strip the last four bytes of a received frame.
[3]	Promiscuous	When set to '1', this bit will cause the RX_MAC to accept all valid frames from the network, regardless of the contents of the DA field of a frame.
[4]	Promiscuous_Group	When set to '1', this bit will cause the RxMAC to accept all valid multicast frames (frames that have the "group" bit in the DA field set to '1').
[5]	Hash_Filter_Enable	When set to '1', the RX_MAC will use the Hash Table to filter multicast addresses

Table 11-38 RX_MAC Configuration Register

Bits	Field Name	Description
[6]	Address_Filter_Enable	When set to '1', this bit will cause the RxMAC to use the Address Filtering Registers to filter both unicast and multicast addresses.
[7]	Disable_Discard_on_Err	When set to '1', this bit will cause the RxMAC to pass errored frames to the RX DMA by setting the BAD bit but not the Abort bit in the status. Frame validity checking for CRC, framing and length errors will not increment error counters. Frames which don't match on destination address will be passed up with the BAD bit set.
[8]	RX_Carrier_Extension	When set to '1', this bit enables the receive part of the Carrier Extension feature. This feature enables the reception of packet bursts generated by Carrier Extension with Packet Bursting senders. This mode only applies for half-duplex at 1Gbps. This bit should be cleared otherwise.

Note: When the CRC is not stripped, reassembly packets will not contain the CRC. These will be stripped by the HRP module because it is reassembling Layer 4 data, and the CRC is Layer 2. However, non-reassembly packets will still contain the CRC when passed to the host.

Programming Restrictions:

1. To ensure proper operation of the RX_MAC, the RX_MAC_Enable bit must always be cleared to '0' and a delay of 3.2msec imposed before a PIO write to any of the other bits in the RX_MAC Configuration register or any of the MAC parameters' registers is performed. To avoid the requirement for a fixed time delay, the RX_MAC_Enable bit may be polled, and when this bit reads back as a '0', all the registers mentioned above may be written, including other bits in the Configuration register.

2. To ensure proper operation of the RX_MAC, the Hash_Filter_Enable bit in the RX_MAC Configuration register must always be cleared to '0' and a delay of 3.2msec imposed before a PIO write to any of the Hash Table registers is performed. To avoid the requirement for a fixed time delay, the Hash_Filter_Enable bit may be polled, and when this bit reads back as a '0', all the registers mentioned above may be written.

3. To ensure proper operation of the RX_MAC, the Address_Filter_Enable bit in the RX_MAC Configuration register must always be cleared to '0' and a delay of 3.2msec imposed before a PIO write to any of the Address Filter registers is performed. To avoid the requirement for a fixed time delay, the

Address_Filter_Enable bit may be polled, and when this bit reads back as a '0', all the registers mentioned above may be written.

MAC Control Configuration Register (RW)

[0]: Send_Pause_Enable: When set to '1', this bit enables the MAC to start responding to requests for sending Pause Flow Control frames.
If cleared to '0', the MAC will ignore both the software Send_Pause commands and the flow control handshake requests on the RxDMA<->MAC interface.

[1]: Receive_Pause_Enable: When set to '1', this bit enables the MAC to start responding to received Pause Flow Control packets.
If cleared to '0', the MAC will ignore all received Pause Flow Control packets.

[2]: Pass_MAC_Control: When set to '1', this bit enables the RxMAC to pass valid MAC Control packets to the RxDMA.
If cleared to '0', the RxMAC will "consume" the MAC Control frame, and will not pass it to RxDMA.

Default: 0x0.

11.1.7.4 Protocol Parameters Registers

Inter Packet Gap 0 Register (RW)

The value in this 8-bit register is used as an extension of the value in the Inter Packet Gap 1 Register for timing the Receive-to-Transmit IPG. The TxMAC will ignore this value and presume it to be equal to zero for the purpose of timing the Transmit-to-Transmit IPG, and/or when the Enable_IPG0 bit in the TxMAC Configuration Register is cleared to '0'.
The time interval specified in this register is in units of media byte time.

Recommended value: 0x00.

Inter Packet Gap 1 Register (RW)

The value in this 8-bit register defines the first 2/3 portion of the Inter Packet Gap (IPG), which is timed by the TxMAC before each frame's transmission is initiated.

For back-to-back transmissions (Transmit-to-Transmit IPG), this value is added to the value in the Inter Packet Gap 2 Register, and during the entire period of time the CarrierSense indication (if present) is ignored by the TxMAC.

For a frame reception followed by a frame transmission (Receive-to-Transmit IPG), the TxMAC will monitor the CarrierSense indication, and (if present) will reset its deferral process during the time interval defined by the sum of values in this register and in the Inter Packet Gap 0 Register, but will ignore it during the time interval specified in the Inter Packet Gap 2 Register.

The time interval specified in this register is in units of media byte time.

Recommended value: 0x08.

Inter Packet Gap 2 Register (RW)

The value in this 8-bit register determines the second 1/3 portion of the Inter Packet Gap parameter.
The time interval specified in this register is in units of media byte time.

Recommended value: 0x04.

Slot Time Register (RW)

This 10-bit register specifies the slot time parameter in units of media byte time. This parameter determines the physical span of the network.

Recommended value: 0x40.

Minimum Frame Size Register (RW)

This 10-bit register determines the minimum number of bytes that the TxMAC will transmit for any frame on the medium, and the RxMAC will receive from the medium before it recognizes the frame to be valid.

Recommended value: 0x40.

Maximum Burst Size/Maximum Frame Size Register (RW)

This 31-bit register specifies two values. Bits [14:0] stores the maximum number of bytes that the TxMAC will transmit for any frame on the medium, and the RxMAC will receive from the medium before it recognizes the frame to be not valid. On bits [29:16], this register stores the MAC Maximum Burst Size.

Recommended value: 0x2000.05EE

PA Size Register (RW)

This 10-bit register specifies the number of Preamble bytes that the TxMAC will transmit at the beginning of each frame. The value programmed in this register should be two or greater.

Recommended value: 0x07.

Jam Size Register (RW)

This 4-bit register specifies the duration of the jam in units of media byte time.

Recommended value: 0x04.

Attempts Limit Register (RW)

This 8-bit register specifies the number of attempts that the TxMAC will make to transmit a frame, before it resets its Attempts_Counter. After reaching the Attempts_Limit, the TxMAC may or may not drop the frame, as determined by the NGU and NGUL bits in the TxMAC Configuration Register.

Recommended value: 0x10.

MAC Control Type Register (RW)

This 16-bit register specifies the "Type" field of a MAC Control frame. The TxMAC uses this field to encapsulate a MAC Control frame for transmission, and the RxMAC uses it for decoding valid MAC Control frames received from the network.

Recommended value: 0x8808.

11.1.7.5 Address Detection and Filtering Registers

MAC Address 0 Register (RW)

This 16-bit register contains the most significant bits of a station's normal priority MAC Address. These bits will be compared against bits [47:32] of the DA field for every frame that arrives from the network. The MAC address programmed in this register must be a unicast address.

MAC Address 1 Register (RW)

This register contains the 16 middle bits of a station's normal priority MAC Address. These bits will be compared against bits [31:16] of the DA field for every frame that arrives from the network. The MAC address programmed in this register must be a unicast address.

MAC Address 2 Register (RW)

This register contains the 16 least significant bits of a station's normal priority MAC Address. These bits will be compared against bits [15:0] of the DA field for every frame that arrives from the network. The MAC address programmed in this register must be a unicast address.

MAC Address 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39 Registers (RW)

This register contains the 16 most significant bits of the station's alternate MAC Addresses 1-15. These bits will be compared against bits [47:32] of the DA field for every frame that arrives from the network. The MAC address programmed in this register may be a unicast or a multicast address.

MAC Address 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, Registers (RW)

This register contains the 16 middle bits of the station's alternate MAC Addresses 1-15. These bits will be compared against bits [31:16] of the DA field for every frame that arrives from the network. The MAC address programmed in this register may be a unicast or a multicast address.

MAC Address 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41 Register (RW)

This register contains the 16 least significant bits of the station's alternate MAC Addresses 1-15. These bits will be compared against bits [15:0] of the DA field for every frame that arrives from the network. The MAC address programmed in this register may be a unicast or a multicast address.

MAC Address 42 Register (RW)

This register contains the 16 most significant bits of the MAC Control Address. These bits will be compared against bits [47:32] of the DA field for every frame that arrives from the network. The MAC address programmed in this register must be the reserved multicast address for MAC Control frames. If there is a match, the MAC will set the bit for Alternative Address Filter Pass [15].

MAC Address 43 Register (RW)

This register contains the 16 middle bits of the MAC Control Address. These bits will be compared against bits [31:16] of the DA field for every frame that arrives from the network. The MAC address programmed in this register must be the reserved multicast address for MAC Control frames. If there is a match, the MAC will set the bit for Alternative Address Filter Pass [15].

MAC Address 44 Register (RW)

This register contains the 16 least significant bits of the MAC Control Address. These bits will be compared against bits [15:0] of the DA field for every frame that arrives from the network. The MAC address programmed in this register must be the reserved multicast address for MAC Control frames. If there is a match, the MAC will set the bit for Alternative Address Filter Pass [15].

Note – A MAC Address of the form a:b:c:d:e:f where bytes **abc** comprise the OUI part of the address and bytes **def** are the vendor unique serial number would be stored as shown Table 11-39. The Group Address bit is the least significant bit of byte **a**.

Table 11-39 MAC Address Notation a:b:c:d:e:f

	ab	**cd**	**ef**
Station Unique Address	MAC Address 2 Register	MAC Address 1 Register	MAC Address 0 Register
Alternate Address 1	MAC Address 5 Register	MAC Address 4 Register	MAC Address 3 Register
MAC Control Address	MAC Address 44 Register	MAC Address 43 Register	MAC Address 42 Register

Address Filter 0 Register (RW)

This 16-bit register contains bits [47:32] of the Address Filter.

Address Filter 1 Register (RW)

This 16-bit register contains bits [31:16] of the Address Filter.

Address Filter 2 Register (RW)

This 16-bit register contains bits [15:0] of the Address Filter.

Address Filter 2&1 Mask Register (RW)

This 8-bit register contains a nibble mask for Address Filter Registers 2 and 1.

Address Filter 0 Mask Register (RW)

This 16-bit register contains a bit mask for the Address Filter Register 0.

Hash Table 0 Register (RW)

This 16-bit register contains bits [255:240] of the Hash Table.

Hash Table 1 Register (RW)

This 16-bit register contains bits [239:224] of the Hash Table.

Hash Table 2 Register (RW)

This 16-bit register contains bits [223:208] of the Hash Table.

Hash Table 3 Register (RW)

This 16-bit register contains bits [207:192] of the Hash Table.

Hash Table 4 Register (RW)

This 16-bit register contains bits [191:176] of the Hash Table.

Hash Table 5 Register (RW)

This 16-bit register contains bits [175:160] of the Hash Table.

Hash Table 6 Register (RW)

This 16-bit register contains bits [159:144] of the Hash Table.

Hash Table 7 Register (RW)

This 16-bit register contains bits [143:128] of the Hash Table.

Hash Table 8 Register (RW)

This 16-bit register contains bits [127:112] of the Hash Table.

Hash Table 9 Register (RW)

This 16-bit register contains bits [111:96] of the Hash Table.

Hash Table 10 Register (RW)

This 16-bit register contains bits [95:80] of the Hash Table.

Hash Table 11 Register (RW)

This 16-bit register contains bits [79:64] of the Hash Table.

Hash Table 12 Register (RW)

This 16-bit register contains bits [63:48] of the Hash Table.

Hash Table 13 Register (RW)

This 16-bit register contains bits [47:32] of the Hash Table.

Hash Table 14 Register (RW)

This 16-bit register contains bits [31:16] of the Hash Table.

Hash Table 15 Register (RW)

This 16-bit register contains bits [15:0] of the Hash Table.

11.1.7.6 Statistics Registers

These counters generate an interrupt upon overflow in the TxMAC Status Register.

Normal Collision Counter (RW)

This 16-bit loadable counter increments for every frame transmission attempt that resulted in a collision.

Recommended initialization to: 0x0000.

First Attempt Successful Collision Counter (RW)

This 16-bit loadable counter will increment for every frame transmission that experienced a collision on the first attempt, but was successfully transmitted on the second attempt.

Recommended initialization to: 0x0000.

Excessive Collision Counter (RW)

This 16-bit loadable counter increments for every frame transmission that has exceeded the Attempts Limit.

Recommended initialization to: 0x0000.

Late Collision Counter (RW)

This 16-bit loadable counter increments for every frame transmission that has experienced a late collision. It indicates the number of frames that the TxMAC has dropped due to collisions that occurred after it has transmitted at least the Minimum Frame Size number of bytes. Usually this is an indication that there is at least one station on the network that violates the maximum allowed span of the network.

Recommended initialization to: 0x0000.

Defer Timer (RW)

This 16-bit loadable timer increments when the TxMAC is deferring to traffic on the network while it is attempting to transmit a frame. The time base for the timer is the media byte clock divided by 256. Thus, on a 10Mbps network the timer ticks are 200 *m*sec, and on a 100Mbps network the timer ticks are 20*m*sec. Software may clear it by writing zeros to the timer.

Recommended initialization to: 0x0000.

Peak Attempts Register (R-AC)

This 8-bit register indicates the highest number of consecutive collisions per successfully transmitted frame, that have occurred since this register was last read. The maximum value that this register can attain is 255. A maskable interrupt is generated to the software if the number of consecutive collisions per successfully transmitted frame exceeds 255. This register will automatically be cleared to '0' after it is read.
Recommended initialization to: 0x00.

Receive Frame Counter (RW)

This 16-bit loadable counter increments after a valid frame has been received from the network.

Recommended initialization to: 0x0000.

Length Error Counter (RW)

This 16-bit loadable counter increments after a frame, whose length is greater than the value that was programmed in the Maximum Frame Size Register, has been received from the network.

Recommended initialization to: 0x0000.

Alignment Error Counter (RW)

This 16-bit loadable counter increments when an alignment error was detected in a receive frame. An alignment error is reported when a receive frame fails the CRC checking algorithm, AND the frame contains a non-integer number of bytes (i.e. the frame size in bits modulo 8 is not equal to zero).

Recommended initialization to: 0x0000.

FCS Error Counter (RW)

This 16-bit loadable counter increments when a receive frame fails the CRC checking algorithm, AND the frame contains an integer number of bytes (i.e. the frame size in bits modulo 8 is equal to zero).

Recommended initialization to: 0x0000.

Rx Code Violation Counter (RW)

This 16-bit loadable counter increments when a Rx_Err indication is generated by the XCVR over the MII, while a frame is being received. This indication is generated by the transceiver when it detects an invalid code in the received data stream. A receive code violation is not counted as an FCS or an Alignment error.

Recommended initialization to: 0x0000.

11.1.7.7 Miscellaneous Registers

Random Number Seed Register (RW)

This 10-bit register is used as a seed for the random number generator for the CSMA/CD backoff algorithm. The register has significance only after power-on reset, and should be programmed with a random value which has a high likelihood of being unique for each MAC attached to a network segment (e.g. 10 LSB of the MAC address). During normal operation, the register contents are constantly updated by the hardware with new "random" values. Therefore, a PIO read from this register will return an unpredictable result.

Pause Timer (RO)

This 16-bit timer is used to time the pause interval as indicated by a received pause flow control frame. A non-zero value in this timer indicates that the MAC is currently in the paused state.

Default: 0x000.

MAC State Machine Register (RO)

This 27-bit register provides the current state for key state machines in the MAC.

11.1.7.8 MIF Programmable Resources

MIF Bit-Bang Clock (RW)

This 1-bit register is used to generate the MDC clock waveform on the MII Management Interface when the MIF is programmed in the "Bit-Bang" Mode. Writing a '1' after a '0' into this register will create a rising edge on the MDC, while writing a '0' after a '1' will create a falling edge. For every bit that is transferred on the management interface, both edges have to be generated.

MIF Bit-Bang Data (RW)

This 1-bit register is used to generate the outgoing data (MDO) on the MII Management Interface when the MIF is programmed in the "Bit-Bang" Mode. The data will be steered to the appropriate MDIO based on the state of the PHY_Select bit in the MIF Configuration Register.

MIF Bit-Bang Output Enable (RW)

This 1-bit register is used to enable ('1') and disable ('0') the I-directional driver on the MII Management Interface when the MIF is programmed in the "Bit-Bang" Mode. The MDIO should be enabled when data bits are transferred from the MIF to the transceiver, and it should be disabled when the interface is idle or when data bits are transferred from the transceiver to the MIF (data portion of a read instruction). Only one MDIO will be enabled at a given time, depending on the state of the PHY_Select bit in the MIF Configuration Register.

MIF Configuration Register (RW)

This 15-bit register controls the operation of the MIF.

Table 11-40 MIF Configuration Register

Bits	Field Name	Description
[0]	PHY_Select	The MIF implements two independent management interfaces for two separate transceivers. Only one transceiver can be used at a given time. This bit determines which transceiver is currently in use. When cleared to '0', MDIO_0 is selected. Went set to '1', MDIO_1 is selected.
[1]	Poll_Enable	When set to '1', this bit enables the polling mechanism as described in 2.4.6. If this bit is set to '1', the BB_Mode should be cleared to '0'
[2]	BB_Mode	This bit determines the mode of operation of the MIF. When set to '1', the "bit-bang mode" is selected. When cleared to '0', the "frame mode" will be used.
[7:3]	Poll_Reg_Addr	This field determines the register address in the transceiver that will be polled by the polling mechanism in the MIF. It is meaningful only if the Poll_Enable bit is set to '1'

Table 11-40 MIF Configuration Register

Bits	Field Name	Description
[8]	MDI_0	This read-only bit is dual-purpose. **When the MDIO_0 interface is idle**, this bit will indicate whether a transceiver is connected to this line. If this bit reads as '1', the transceiver is connected. When the MIF is communicating with a transceiver that is hooked up to MDIO_0 in the bit-bang mode, this bit will indicate the incoming bit stream during a read operation
[9]	MDI_1	This read-only bit is dual-purpose. **When the MDIO_1 interface is idle**, this bit will indicate whether a transceiver is connected to this line. If this bit reads as '1', the transceiver is connected. When the MIF is communicating with a transceiver that is hooked up to MDIO_1 in the bit-bang mode, this bit will indicate the incoming bit stream during a read operation
[14:10]	Poll_Phy_Addr	This field determines the transceiver address to be polled

MIF Frame/Output Register (RW)

This 32-bit register serves as an "Instruction Register" when the MIF is programmed in the Frame Mode. In order to execute a read/write operation from/to a transceiver register, the software has to load this register with a valid instruction, as per IEEE 802.3 MII specification. After issuing an instruction, the software has to poll this register to check for instruction execution completion. During a read operation, this register will also contain the 16-bit data that was returned by the transceiver.

Table 11-41 MIF Frame/Output Register

Bits	Field Name	Description
[31:30]	ST	STart of frame. When issuing an instruction: This field should always be loaded with a '01'. When polling for completion: This field is always a "don't care".
[29:28]	OP	OPcode. When issuing an instruction: This field should be loaded with '01' for a "write" and with '10' for a "read". When polling for completion: This field is always a "don't care".
[27:23]	PHYAD	PHY ADdress. When issuing an instruction: This field should be loaded with the XCVR address. When polling for completion: This field is always a "don't care".

Table 11-41 MIF Frame / Output Register

Bits	Field Name	Description
[22:18]	REGAD	REGister ADdress. When issuing an instruction: This field should be loaded with the address of the register that is to be read / written. When polling for completion: This field is always a "don't care".
[17]	TA_MSB	Turn Around, Most Significant Bit. When issuing an instruction: This bit should always be loaded with a '1'. When polling for completion: This bit is always a "don't care".
[16]	TA_LSB	Turn Around, Least Significant Bit. When issuing an instruction: This bit should always be loaded with a '0'. When polling for completion: This bit serves as a "Valid Bit". When this bit is set to'1', the instruction execution has been completed.
[15:0]	DATA	Instruction Payload. When issuing an instruction: This field should be loaded with the 16-bit data to be written into a transceiver register for a "write", and is a "don't care" for a "read". When polling for completion: This field is a "don't care" for a "write", and contains the 16-bit data returned by the transceiver for a "read" (if the Valid Bit is set).

MIF Status Register (R-AC)

This 32-bit register is used in conjunction with the Poll Mode in the MIF. It contains two portions: Poll Data and Poll Status. The Poll Data field will always contain the latest "image update" of the XCVR register that is being polled, while the Poll Status field will indicate which bits in the Poll Data field have changed since the MIF Status Register was last read. The Poll Status field is "auto-cleared" after being read.

Table 11-42 MIF Status Register

Bits	Field Name	Description
[31:16]	Poll_Data	
[15:0]	Poll_Status	

MIF Mask Register (RW)

This 16-bit register is used to determine which bits in the Poll Status portion of the MIF Status Register will cause an interrupt. If a mask bit is cleared to '0', the corresponding bit of the Poll Status will generate the MIF Interrupt when set.

Default: 0xFFFF.

MIF State Machine Register (RO)

This 7-bit register provides the current state for all the state machines in the MIF.

Table 11-43 MIF State Machine Register

Bits	Field Name	Description
[2:0]		Control State Machine State
[6:5]		Execution State Machine State

11.1.8 PCS Programmable Resources

PCS MII Control Register (RW)

This register is equivalent to the Control Register of the standard MII management register set. Unlike MII management registers, this register is directly mapped in Cassini's register space.

Table 11-44 PCS MII Control Register

Bits	Field Name	Description
[5:0]	Reserved	
[6]	1000 Mb/s Speed Select	Reads back one. Ignored on writes.
[7]	Collision Test	When set the COL signal at the PCS to MAC interface is activated regardless of activity on the receive inputs.
[8]	Duplex Mode	Forced low. PCS behavior is similar for half and full duplex.
[9]	Restart Auto-Negotiation	Self clearing bit. Set to restart Auto-Negotiation.
[10]	Isolate	Reads back as zero. Ignored on writes.
[11]	Power Down	Reads back as zero. Ignored on writes.
[12]	Auto-Negotiation Enable	When set the PCS goes through an automatic link configuration phase before it can be used. When clear the link can be used without any link configuration phase. Defaults high.
[13]	10/100 Speed Selection	Reads back as zero. Ignored on writes.
[14]	reserved	Reads back as zero. Ignored on writes.
[15]	Reset	Resets the PCS when set. Self-clearing when reset process is complete.

Default: 0x1040.

Note – The Auto-Negotiation Enable bit should be programmed the same at the Link Partner as in the local device. Otherwise, Auto-Negotiation will not complete. When this bit is reprogrammed, Auto-Negotiation/Manual-Configuration is restarted automatically.

PCS MII Status Register (RO)

This register is equivalent to the Status Register of the standard MII management register set. Unlike MII management registers, this register is directly mapped in Cassini's register space.

Table 11-45 PCS MII Status Register

Bits	Field Name	Description
[0]	Extended Capability	Reads zero.
[1]	Jabber Detect	Reads zero.
[2]	Link Status	1=Link is up; 0=Link is down This bit incorporates a latch on 0, such that the 0 is kept until read. The register may be read twice to determine if the link has gone up again.
[3]	Auto-Negotiation Ability	Always reads 1 (Able to perform Auto-negotiation)
[4]	Remote Fault	When set, this bit indicates that a Remote Fault has been detected from the received Link Code Word. Only valid after completion of Auto-Negotiation.
[5]	Auto-Negotiation Complete	1=Auto-Negotiation complete 0=Auto-Negotiation not completed
[7:6]	Reserved	
[8]	Extended Status	This bit can be used as an indication that this is a 1000 Base-X PHY. Reads back one. Writes to it are ignored.
[15:9]	Reserved.	

Default: 0x0108.

PCS MII Advertisement Register (RW)

This register is equivalent to the Advertisement Register of the standard MII management register set. Unlike MII management registers, this register is directly mapped in Cassini's register space. The contents of this register are used during Auto-Negotiation.

Table 11-46 PCS MII Advertisement Register

Bits	Field Name	Description
[4:0]	Reserved	
[5]	Full Duplex	Advertises device's capability to perform full duplex 1000 Base-X.
[6]	Half Duplex	Advertises device's capability to perform half duplex 1000 Base-X.
[7]	PAUSE	Advertises device's capability to perform PAUSE symmetrical to its link partner.
[8]	ASM_DIR	Advertises device's capability to perform PAUSE asymmetric to its link partner.

Table 11-46 PCS MII Advertisement Register

Bits	Field Name	Description
[11:9]	Reserved	
[13:12]	Remote Fault	Provides simple fault information to the link partner. Bit 13 is writable by software to optionally indicate to its link partner that it is going off-line. Bit 12 will get set when signal_detect equals FAIL, and will remain set until successful negotiation at which time it will be set to zero.
[14]	Ack	Read Only.
[15]	Next Page	Read Only. Forced low.

Default: 0x00E0.

PCS MII Link Partner Ability Register (RW)

This register is equivalent to the Link Partner Ability Register of the standard MII management register set. Unlike MII management registers, this register is directly mapped in Cassini's register space. The contents of this register are updated as a result of Auto-Negotiation. The register layout and bit definitions correspond to the PCS MII Advertisement Register.

PCS Configuration Register (RW)

Table 11-47 PCS Configuration Register

Bits	Field Name	Description
[0]	Enable	Enables the PCS functions. Must be zero while modifying the PCS MII Advertisement Register.
[1]	Signal_detect override	Sets signal_detect internally to OK in case the optic is not able to generate a reliable signal. This bit is non-resettable so that it can mimic a pullup or pulldown on the board.
[2]	Signal_detect active low	When set, changes the interpretation of the incoming optic signal so that when the signal is low, signal_detect is OK.
[4:3]	jitter_study	Allows detailed jitter measurements to be made to characterize optic parts. A single code group is transmitted repeatedly. Disparity rules are followed. Program to zero for normal operation. 0x0 = normal operation 0x1 = high frequency test pattern, D21.5 0x2 = low frequency test pattern, K28.7 0x3 = reserved
[5]	10 ms timer override	Shortens the 10-20 ms timer used in Auto-Negotiation and synchronization to a few cycles. The purpose is to shorten ASIC simulation time and vector generation. This should be programmed to zero for interoperability with other devices.

Default: 0x0.

PCS State Machine Register (RO)

Table 11-48 PCS State Machine Register

Bits	Field Name	Description
[3:0]	Tx control state	States 0 and 1 indicate transmission of idle. Otherwise it indicates transmission of a packet.
[7:4]	Rx control state	State 0 indicates reception of idle. Otherwise it indicates reception of a packet.
[10:8]	Word synchronization state	State 0 indicates loss of sync. Otherwise sync has been acquired.
[12;11]	Sequence detection state	Cycling through states 0-3 indicates reception of Config codes. Cycling through states 0-1 indicates reception of idles.
[16:13]	Link configuration state	State of 0xB indicates link is up.
[19:17]	Reserved	
[20]	Loss of link due to C codes	Gets set to 1 if receipt of Configuration codes from th link partner caused loss of link.
[21]	Loss of link due to loss of sync	Gets set to 1 if loss of sync caused loss of link.
[22]	Loss of signal_detect	Gets set to 1 if signal_detect goes from OK to FAIL. This will cause loss of link and loss of sync. If this bit is set, bit 29 will also be set.
[23]	reserved	
[24]	Link not up due to receipt of breaklink C codes from partner.	Set to '1' when C codes with contents of zero are received triggering start or restart of autonegotiation. They should be sent for no longer than 20 ms.
[25]	Link not up due to Serdes.	Set to '1' when the Serdes is being initialized. See Serdes state register. It is possible that the Serdes is not providing SYNC_DET signals indicating that it has achieved byte synchronization.
[26]	Link not up due to lack of good C codes.	Set to '1' when the contents of the C codes is not stable or C codes are not being received.
[27]	Link not up due to loss of sync.	Set to '1' when word sync has not been achieved. This can occur due to bit errors, receipt of illegal codes or comma characters clocking on RBC0.
[28]	Link not up because waiting for C codes with ack bits set.	Set to '1' when waiting for C codes with acknowledge bit set indicating Link partner has received our C codes.
[29]	Link not up because partner fails to send idle.	Set to '1' when at the end of autonegotiation the Link Partner continues to send C codes instead of idle symbols or packet data.
[31:30]	Reserved	

These bits are diagnostic in nature. Bits 20 to 22 are auto-clear on read.

Default: 0x00000

PCS Interrupt Status Register (R-AC)

This register indicates interrupt changes in specific PCS MII Status bits. Presently only one bit is implemented, reflecting transitions on the link status. The rationale for a single bit is that all other relevant bits cannot change without altering the link state. There is no mask register at this level. The PCS_INT bit may be masked at the Interrupt Status Register level.

Table 11-49 PCS Interrupt Status Register

Bits	Field Name	Description
[2]	Link Status Change	When set it indicates the Link Status has changed at least once since this register was read.

11.1.9 Serialink Programmable Resources

Datapath Mode Register (RW)

This register controls which network interface is used. No more than one bit should be set in this register.

Table 11-50 Datapath Mode Register

Bits	Field Name	Description
[0]	MII/GMII and MAC loopback Mode	When set to '1' the PCS is not used, and the MII/GMII interface and clocking is selected. Selection between MII and GMII is also controlled by the XIF register.
[1]	External SERDES Mode	When set to '1' the PCS is used via the 10-bit interface.

Default: none.

Serdes Control Register (RW)

This register controls inputs to the Serdes chip or Serialink block.

Table 11-51 Serdes Control Register

Bits	Field Name	Description
[0]	LOOPBACK	Loopback is enabled at the 10 bit interface of the external Serdes when this bit is set.
[1]	ENSYNCDET	Enable Sync Character Detection. Setting this bit enables the receiver to detect the sync character in the received data. When clear sync character detection is automatically disabled once the receiver establishes synchronization. Should be low for normal operation.
[2]	LOCKREF	Lock to reference clock. When set, it causes the receiver to frequency-lock RBC[0:1] to the reference clock REFCLK. When clear, the receiver's clock generator locks to incoming serial data.

Default: 0x000

Shared Output Select Register (RW)

This register controls the multiplexing of test outputs into the PROM address (PA_3 through PA_0) pins. Should be set to 0x0 for normal operation.

Table 11-52 Datapath Mode Register

Bits	Field Name	Description
[2:0]	PROM_Addr	000 - Normal operation, PROM addresses [3:0] are selected. 001 - {rxdma_req, rxdma_ack, rxdma_rdy, rxdma_rd} are selected. 010 - {rx_mac_req, rx_ack, rx_tag, rxclk_shared} are selected. 011 - {tx_mac_req,tx_ack,tx_tag,tx_retry_req}. 100 - {tx_tp3,tx_tp2,tx_tp1,tx_tp0} 101 - {R_period_rx,R_period_tx, R_period_hp,R_period_bim}

Default: 0x0.

Serdes State Register (RO)

This register indicates the progress of the Serdes boot up. Used for diagnostic purposes, may be ignored by normal software drivers.

Table 11-53 Datapath Mode Register

Bits	Field Name	Description
[1:0]	Serdes State	00- Undergoing reset 01- Waiting 500us while lockrefn is asserted 10- Waiting for comma detect 11- Receive data is synchronized

Default: 0x0.

PCS Packet Counter (RO)

This register indicates the number of packets transmitted or received. Used for diagnostic purposes, may be ignored by normal software drivers. The counters

Table 11-54 PCS Packet Counter Register

Bits	Field Name	Description
[10:0]	tx_pkt_cnt	Number of packets transmitted by the PCS.
[15:11]	Reserved	
[26:16]	rx_pkt_cnt	Number of packets received by the PCS whether they encountered an error or not.
[32:27]	Reserved	

roll over without generating an interrupt when they reach their terminal count.

Default: 0x0.

11.1.10 PROM Image

The PROM image mapped in Cassini's register space provides an additional mechanism for reading the PROM, and unlike the space programmed in the Expansion ROM Base Address Register, it may be used by the software driver at any time.

11.2 Programming Notes

11.2.1 Initialization Sequence

This section describes Cassini's software initialization sequence for typical environments. Register initialization consists mostly of writing constants to the relevant registers, except for a few cases where the value to be written is determined during run-time as a function of:

1. host system characteristics / driver configuration files
2. values read from other Cassini registers
3. values read from PHY devices and interface pins

The basic initialization flow is shown in Figure 11-1, and the detailed guidelines for each step are discussed below. This flow assumes a full blown Cassini environment (NIC or motherboard) including at least one PHY device. Simpler diagnostic environments might omit some of the steps.

A) After hardware reset, Cassini's PCI Configuration Space must be initialized. This is usually done by the platform OBP or BIOS function, with no driver involvement. This process must initialize the following Cassini PCI Configuration Space Registers:

- Command Register - Bits that must be enabled: Memory Space, Bus Master. Recommended if possible: MWI Enable, Parity Error Response.
- Cache Line Size.
- Base Address Register.
- Interrupt Line - Not interpreted by Cassini, but this information might be needed by the driver.

B) At this point Cassini's register space is accessible to the driver. The Random Number Seed Register should be written at this time with a 10 bit value uncorrelated across different nodes.

C) Proceed to program the physical layer registers:

- Datapath Mode Register - Set to either Serdes, or MII / GMII mode.
- If MII, determine management interface to use (MDIO1 corresponds to external transceiver and has precedence over MDIO0), take PHY out of isolate mode, and initialize PHY. Enable MII PHY Auto-negotiation.

D) Set up the TX and RX DMA configuration by programming:

- TX Descriptor Ring Size and Base -
- TX FIFO Threshold -
- RX Free and Completion Descriptor Ring Sizes and Base Addresses -
- Pause Thresholds -
- RX Blanking -
- RX Almost Empty thresholds
- RX Page configuration register
- RX Random Early Detect Probabilities
- Rx Parser Configuration Register

- Initialize the Rx Batching Table so the values are defined (needed for simulation only). Leave the control address register at zero before enabling the Rx.
- Initialize the Rx Reassembly DMA Table so the valid bits are zero for all 64 entries.
- Initialize the Header Parser Instruction RAM with desired microcode.
- Rx Configuration register. Enable Rx after other Rx registers are initialized.
- Enable Tx.

Note – For software compatibility with future variants of Cassini, the OFF threshold and ON threshold in the PAUSE Thresholds Register should be programmed considering the value read from the RX FIFO Size Register. For lossless flow control the OFF threshold should be between 4.5 to 6kbytes below the RX Fifo Size. This value depends on the link length. It is possible to make the OFF threshold adaptive starting with an arbitrary value and lowering the threshold if RX FIFO Overflows occur. A simple way of disabling PAUSE frame emission is to program the OFF threshold to the full RX FIFO Size.

E) Set up the MAC by programming a sane set of values. Note that most MAC registers power up undefined. Recommended values for these MAC registers are given in Table 11-55. It is important to achieve a stable clocking within the MAC prior to initialization of all MAC registers. Consequently, program the XIF register first followed by MAC rx/tx software reset, followed by the initialization of the remaining MAC registers.

F) Initialize and enable the PCS if Serdes GMII mode is selected. Wait for Link UP.

If Datapath is set for MII wait for MII transceiver link up. If there is no MII link but there is PCS link, go back to C) with a revised Datapath value.

If Datapath was not set for MII, wait until PCS link goes up.

The critical link negotiated values are only valid when the link is up (Auto-negotiation succeeded). Either stay here until link is up, or enable the appropriate Link status change interrupt and wake up when it is done.

Verify link is still up, enable interrupt to detect if it goes down. Save the values of:

- Full Duplex -
- Flow Control -
- Carrier Extension -
- Speed -

G) Re-program MAC to match the values found in F). This involves registers:

- TX_MAC Configuration Register - Bits: Ignore_Carrier_Sense, Ignore_Collisions.
- MAC Control Configuration Register - Bits: Send_Pause_Enable, Receive_Pause_Enable.
- Slot Time Register - 64 bytes normally. 512 bytes for half duplex @1Gbps. speed.

- XIF Configuration Register- set 1 for Gbps, clear otherwise, clear MII_Int_loopback.

H) Enable:

- TX Configuration Register - Set the Tx_DMA_Enable bit. Other bits not modified.
- RX Configuration Register - Set the Rx_DMA_Enable bit. Other bits not modified.
- TX_MAC Configuration Register - Set the TxMAC_Enable bit. Other bits not modified.
- RX_MAC Configuration Register - Set the RxMAC_Enable bit. Other bits not modified.
- Interrupt Mask Register - Clear all bits corresponding to desired interrupts, possible bits to enable: TX_ALL, RX_DONE, Rx_Buffer_Not_Available, TX_MAC_INT, RX_MAC_INT, and either the MIF_Interrupt or PCS_INT (depending on PHY interface used).

If the software driver needs to reset Cassini, it should set both the TX and RX Software Reset bits in the Software Reset Register, poll the register until both bits read low, and proceed to step C).

On Link Down the driver should disable the MAC and go to step F).

Note – The MAC should be disabled upon Link Down as soon as possible to minimize the chances of the link going up again without updating the MAC configuration accordingly.

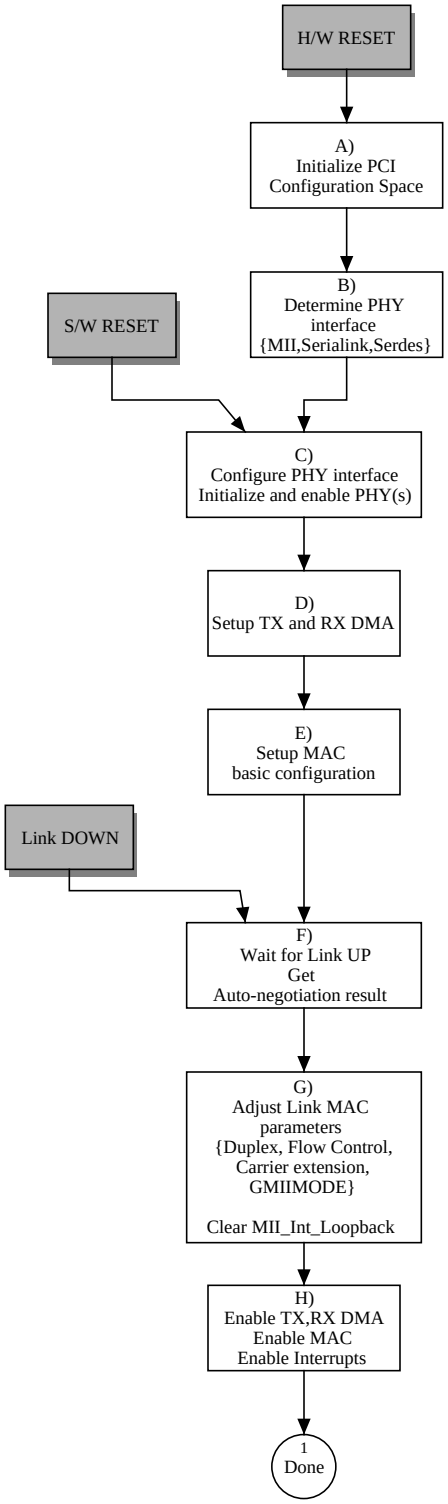


Figure 11-1 Initialization Flow

Table 11-55 MAC recommended initialization values

Address Offset	Initialization Value recommended	Actual Size (bits)	Register	Comment
MAC Registers				
0x6008	0x01BF0	17	Send Pause Command Register	PAUSE value large enough to get DMA moving again, small enough to block the link no worse than 16 retries on half duplex. Can use smaller values @10/100 Mbps.
0x6040	0x00	8	InterPacketGap0 Register	
0x6044	0x08	8	InterPacketGap1 Register	Per 802.3
0x6048	0x04	8	InterPacketGap2 Register	Per 802.3
0x604C	0x040	10	SlotTime Register	Per 802.3, change value later if needed for 802.3z carrier extension
0x6050	0x040	10	MinFrameSizeRegister	Per 802.3
0x6054	0x05EE	15	MaxFrameSize Register	Per 802.3
0x6058	0x07	10	PA Size Register	Per 802.3
0x605C	0x4	4	JamSize Register	Per 802.3
0x6060	0x10	8	Attempt Limit Register	Per 802.3
0x6064	0x8808	16	MAC Control Type Register	Per 802.3x
0x6080	Unique Station Address	16	MAC Address 0 Register	lower serial number bytes
0x6084		16	MAC Address 1 Register	lowest OUI byte, upper serial number byte
0x6088		16	MAC Address 2 Register	upper OUI bytes
See table 3-4	0x0000	16	MAC Address 3,6,9,12,15,18,21,24,27,30,33,36,39 Register	zero until used
See table 3-4	0x0000	16	MAC Address 4,7,10,13,16,19,22,25,28,31,34,37,40 Register	
See table 3-4	0x0000	16	MAC Address 5,8,11,14,17,20,23,26,29,32,35,38,41 Register	
0x6128	0x0180	16	MAC Address 42 Register	0x0001 per 802.3x
0x612C	0xC200	16	MAC Address 43 Register	0xC200 per 802.3x
0x6130	0x0001	16	MAC Address 44 Register	0x0180 per 802.3x

Table 11-55 MAC recommended initialization values

Address Offset	Initialization Value recommended	Actual Size (bits)	Register	Comment
0x614C	0x0000	16	Address Filter 0 Register	zero until used
0x6150	0x0000	16	Address Filter 1 Register	
0x6154	0x0000	16	Address Filter 2 Register	
0x615C	0x00	8	Address Filter 2&1 Mask Register	
0x6160	0x0000	16	Address Filter 0 Mask Register	
0x6060-0x619C	0x0000	16	Hash Table 0 through 15 Registers	zero until used
0x61A0	0x0000	16	Normal Collision Counter	Clear all MAC counters
0x61A4	0x0000	16	First Attempt Successful Collision Counter	
0x61A8	0x0000	16	Excessive Collision Counter	
0x61AC	0x0000	16	Late Collision Counter	
0x61B0	0x0000	16	Defer Timer	
0x61B8	0x0000	16	Receive Frame Counter	
0x61BC	0x0000	16	Length Error Counter	
0x61C0	0x0000	16	Alignment Error Counter	
0x61C4	0x0000	16	FCS Error Counter	
0x61C8	0x0000	16	RX code Violation Error Counter	
0x61CC	0x0000	16	Random Number Seed	

11.2.2 MAC Operational Modes

The following table indicates how the software driver selects the MAC layer network interface mode, in terms of Full Duplex vs. Half Duplex, and different loopback modes. The bits involved are:

- MII_Int_Loopback - XIF Configuration Register
- Disable_Echo - XIF Configuration Register
- Ignore_Carrier_Sense - TX MAC Configuration Register
- Ignore_Collisions - TX MAC Configuration Register
- No_Backoff - TX MAC Configuration Register
- Wrapback - PCS MII Control Register
- LOCKREF - Serialink Control Register
- LOOPBACK - Serialink Control Register

Table 11-56 Full Duplex and Loopback Configuration

	NoBackoff	IgnoreCollisions	IgnoreCarrierSense	DisableEcho	MII_Int_Loopback	Wrapback, LOCKREF bits	LOOPBACK
Full Duplex	X	1	1	0		0	0
Half Duplex	0	0	0	see note-1 below	0	0	0
Internal Loopback at MII	X	0	0	0	1	0	0
Loopback at PCS	X	0	0	0	0	1	0
Loopback at Serialink	X	0	0	0	0	0	1

Note – 1. The Disable_Echo bit should be enabled in MII or GMII half duplex modes when an external PHY transceiver is used. When set to 1, this bit disables the receive data path during packet transmission by the TxMAC. Effectively, the MII/GMII RX_DV signal is forced to 0 when the TX_EN is active. This bit should be cleared to 0 when the MAC operates in any full duplex mode, when it operates in any loopback mode (internal or external) or is in half duplex Serdes or Slink modes.

Pinout and Mechanical 12

12.1 Functional I/Os Description

12.1.1 PCI Interface Signals

PCI_CLK

1

This clock provides the timing for all transactions on PCI bus and is an input to every PCI device. All PCI signals, except RST# and INTA# are sampled on the rising edge of this clock and all timing parameters are defined with respect to this edge. This clock runs up to 66 MHz.

RST#

This active low reset is used to bring PCI-specific registers, sequencers, and signals to a consistent state. Anytime RST# is asserted, all PCI output signals must be driven to their benign state. RST# may be asynchronous to CLK when asserted or de-asserted.

AD[31:0]

Multiplexed Address and Data. A bus transaction consists of an address phase followed by one or more data phases. The address phase is the clock cycle in which FRAME# is asserted.

AD[63:32]

Upper 32 bits of Address and Data bus. During the address phase, the upper 32-bits of the 64-bit address are transferred. Otherwise these bits are reserved. During the data phase, an additional 32-bits of data are transferred when REQ64# and ACK64# are both asserted.

C/BE[3:0]#

Bus Command and Byte Enables are multiplexed on the same PCI pins. During the address phase of a transaction C/BE[3:0]# define the bus command and during data phase they are used as byte enables. The Byte Enables are valid for the entire data phase and determine which byte lanes carry meaningful data. C/BE[0]# applies to byte 0 and C/BE[3]# applies to byte 3.

C/BE[7:4]#

Bus Command and Byte Enables used for 64 bit transfers. During the address phase, the actual command is transferred on C/BE[7:4]#. Otherwise, these bits are reserved. During a data phase, C/BE[7:4]# indicate which byte lanes carry meaningful data when REQ64# and ACK64# are both asserted. C/BE[4]# applies to byte 4 and C/BE[7]# applies to byte 7.

PAR

This signal provides even parity across AD[31:0] and C/BE[3:0]. PAR is stable and valid one clock after the address phase. For data phase, PAR is stable and valid one clock after either IRDY# is asserted on a write transaction or TRDY# is asserted on a read transaction. Once PAR is valid, it remains valid until one clock after the completion of the current data phase. The master drives PAR for address and write data phase; the target drives PAR for read data phase.

PAR64

This signal provides even parity across AD[63:32] and C/BE[7:4]#. PAR64 is valid one clock after the initial address phase when REQ64# is asserted and the DAC command is indicated on C/BE[3:0]#. PAR64 is valid one clock after the second address phase of a DAC command when REQ64# is asserted. PAR64 is stable and valid for data phases when both REQ64# and ACK64# are asserted and one clock after either IRDY# is asserted on a write transaction or TRDY# is asserted on a read transaction. Once PAR64 is valid, it remains valid for one clock after the completion of the data phase.The master drives PAR64 for address and write data phase; the target drives PAR64 for read data phase.

FRAME#

This signal is driven by the current master to indicate the beginning and the duration of an access. FRAME# is asserted to indicate that a bus transaction has started. While FRAME# is asserted, the data transfer continues. When FRAME# becomes de-asserted, the transaction is in the final data phase or has been completed.

IRDY#

The Initiator Ready signal indicates the bus master’s ability to complete the current data phase of transaction. IRDY# is used in conjunction with TRDY#. A data phase is completed on any clock both IRDY# and TRDY# are sampled as asserted. During a write cycle, IRDY# indicates that valid data is present on the AD[31:0] lines. During a read cycle, it indicates that the master is prepared to accept data. Wait cycles are inserted until both IRDY# and TRDY# are asserted together.

TRDY#

The Target Ready signal indicates the selected device's ability to complete the current data phase of transaction. TRDY# is used in conjunction with IRDY#. A data phase is completed on any clock both TRDY# and IRDY# are sampled asserted. During a read cycle, TRDY# indicates that valid data is present on the AD[31:0] lines. During a write cycle, it indicates that the target is prepared to accept data. Wait cycles are inserted until both IRDY# and TRDY# are asserted together.

STOP#

This signal indicates that the current target is requesting the master to stop the current transaction.

IDSEL

The Initialization Device Select signal is used as a "chip select" signal during configuration read and write transactions.

DEVSEL#

The Device Select signal, when actively driven, indicates the driving device has decoded its address as the target of the current access. As an input, DEVSEL# indicates whether any device on the bus has been selected.

REQ#

The Request signal is one of the signals monitored by the arbiter. It indicates that an agent (master) desires the use of the bus. This is a point-to-point signal. Every master has its own dedicated REQ#, which must be tri-stated while RST# is asserted.

REQ64#

This signal, when actively driven by a given bus master, indicates that this master desires to transfer data using 64-bit transactions. REQ64# has the same timing as FRAME#.

GNT#

The arbiter generates this signal, indicating to the master that access to the bus has been granted. This is a point-to-point signal. Every master has its own dedicated GNT#, which must be ignored while RST# is asserted.

ACK64#

Input to Cassini+ . When actively driven by the device that has positively decoded its address as the target of the current access, indicates that the target is willing to transfer data using 64-bit transactions. ACK64# has the same timing as DEVSEL#.

PERR#

This signal is used for reporting data errors during all PCI transactions, except for a Special Cycle. The PERR# pin is sustained tri-state and must be driven active by the agent receiving data, two clocks following the data itself, if a data parity error is detected.

SERR#

The System Error output signal is used for reporting address errors, data parity errors on the Special Cycle command, or any other system error where the result will be “catastrophic”.

INT{A,B,C,D}#

The assertion and de-assertion of INTA# is asynchronous to the PCI_CLK. Used to requesting attention from the device driver. Once the INTA# signal is asserted, it remains asserted until the device driver clears the pending request.

M66EN

Input pin, when high it indicates that the bus segment operates at 66MHz. When low, the bus operates at 33MHz. The state of this pin is readable via Cassini+ register space.

12.1.2 PCI PLL pins

PLL_BYPASS#

Input. The PCI clock PLL function is bypassed when low. PLL is engaged when this input is high. May be controlled using M66EN to select PLL for EPCI (66 MHz PCI).

PLL_CREF

Analog signal. This terminal connects a 1.0nF Capacitor to GND as noisy filter.

PLL_CK_OUT

Open Drain Output. This is a test pin for PLL Jitter, error analysis.

PLL_VDD

Supply for PCI PLL.

PLL_VSS

VSS for PCI PLL.

12.1.3 GMII/SERDES Interface Signals

This group of signals is used for transceiver interface, using one out three possible interfaces: MII, GMII, and 10-bit SERDES.

CRS

Carrier Sense signal for MII/GMII. When asserted, the Carrier Sense signal indicates activity on the medium. This signal is asynchronous with respect to the MII/GMII clocks.

COL

Collision signal for MII/GMII. When asserted, the Collision signal indicates that a collision occurred on the medium. It remains asserted while the collision condition persist. This signal is asynchronous with respect to the MII/GMII clocks.

RXCLK

The Receive Clock input carries a continuous clock sourced by the transceiver, which provides the timing reference for sampling data, RXDV and RXER signals into Cassini+ in MII/GMII modes. The frequency of this clock is the nibble rate for MII, and the byte rate for GMII (25MHz for 100 Mb/s over MII, 2.5MHz for 10 Mb/s over MII, and 125MHz for 1Gb/p over GMII).

RBC0

This input signal carries one phase of the SERDES receive clock, a 62.5MHz clock whose rising edges latch bytes 1 and 3 of the receive word. Bytes 0 and 2. are latched by a different phase of the receive clock RBC[1]. The receive data byte containing the Comma character (byte 0) is clocked by the rising edge of RBC[1]. RBC [0] and RBC [1] may be stretched during alignment.

RBC1

This input signal represents the second phase of the SERDES receive clock, a 62.5MHz clock whose rising edges latch bytes 0 and 2 of the receive word. RBC [0] and RBC [1] may be stretched during alignment.

RX[3:0]_S

Inputs carrying receive data lines 0 through 3 from the transceiver to the receive port of Cassini+ . For MII/GMII these signals are synchronous to the receive clock, and data is transferred for every clock whenever Receive Data Valid is active.

For 10-bit SERDES modes, these lines are sampled alternately by RBC[0] and RBC[1] rising edges.

RX[7:4]_S

Inputs carrying receive data lines 4 through 7 from the transceiver to the receive port of Cassini+ . For GMII these signals are synchronous to the receive clock, and data is transferred for every clock whenever Receive Data Valid is active. These signals are not used in MII mode.

For 10-bit SERDES modes, these lines are sampled alternately by RBC[0] and RBC[1] rising edges.

RX8_S

Input signal used as Received Data Valid in MII/GMII modes, synchronous to the receive clock. Driven by the transceiver to indicate that it is presenting recovered and decoded data on the receive data lines.

Used as receive data line eight (RX8) in 10-bit SERDES mode.

RX9_S

Input signal used as Received Error in MII/GMII modes. Asserted by the transceiver for one or more receive clock periods to indicate that an error was detected in the frame presently being transferred to Cassini+ 's receive port. This signal is synchronous to the receive clock.

Used as receive data line nine (RX9) in 10-bit SERDES mode.

TXCLK_MII

The Transmit Clock input carries a continuous clock sourced by the transceiver, which provides the timing reference for the transfer of the TXEN and transmit data signals from a Cassini+ in MII/GMII modes. The frequency of this clock is the nibble rate for MII, and the byte rate for GMII (25MHz for 100 Mb/s over MII, 2.5MHz for 10 Mb/s over MII, and 125MHz for 1Gb/s over GMII).

TXCLK_S

This output carries the Transmit Clock as a buffered version of *REFCLK* in SERDES/GMII modes.

In SERDES mode it carries the **TBC** (Transmit Byte Clock) output used to clock transmit data presented on TX[0:9]_S. This clock is derived from REFCLK, and has the same frequency and tolerance as REFCLK (typically 125MHz +/- 100 ppm).

TX[3:0]_S

Outputs carrying transmit data lines 0 through 3 from Cassini+ to the transceiver. In MII/GMII modes for each transmit clock period in which TXEN ia asserted, data is transferred from Cassini+ to the transceiver.

TX[7:4]_S

Outputs carrying transmit data lines 4 through 7 from Cassini+ to the transceiver. In MII/GMII modes for each transmit clock period in which TXEN ia asserted, data is transferred from Cassini+ to the transceiver.

TX8_S

In 10-bit SERDES modes this output represents TX8.

In MII/GMII modes this output carries the TXEN signal from Cassini+ to the transceiver. It is driven by Cassini+ to indicates to the transceiver that it is presenting data on the MII/GMII for transmission. This signal remains asserted while all nibbles/bytes to be transmitted are presented to the MII/GMII. This signal is synchronous to TXCLK_MII.

TX9_S

Output carrying TXER from Cassini+ to the transceiver in MII/GMII modes.

In 10-bit SERDES modes this output represents TX9.

12.1.4 Transceiver Management Interface (MIF) Signals

These signals are used for either MII management and serial EEPROM interface. Up to two interfaces may be connected with their own bi-directional serial data line, while sharing the clock.

MDC

The Management Data Clock is sourced by the Cassini+ MIF port to the XCVR as the timing reference for information on the MDIO0 and MDIO1 signals. This clock has a minimum high and low time of 160nS, and a minimum period of 400nS.

MDIO0

Management Data Input/Output 0 is a bi-directional signal between the transceiver and the Cassini+ MIF interface for serial data transfer. This line carries the control information which is driven by the MIF interface with respect to MDC, and is sampled synchronously by the transceiver. Status information is driven by the transceiver synchronously with respect to the MDC clock, and is sampled synchronously by the MIF interface.

MDIO1

Management Data Input/Output 1 is a bi-directional signal between the transceiver and the Cassini+ MIF interface for serial data transfer. This line carries the control information which is driven by the MIF interface with respect to MDC, and is sampled synchronously by the transceiver. Status information is driven by the transceiver synchronously with respect to the MDC clock, and is sampled synchronously by the MIF interface.

MII_BUF_EN

Output port used to enable MII interface external buffers if more than one interface per board is used. This pin is controlled by the MII_Buffer_OE bit in the *XIF Configuration Register*.

RSTOUT#

Reset output used to reset other board devices (PHYs). This pin is activated (low) whenever the RST# input is low or the RSTOUT bit in the *XIF Configuration Register* is set.

12.1.5 Dedicated SERDES Interface signals

The following signals have no other uses beyond SERDES interface.

EWRAP

Output used for controlling the physical layer loopback operation. When high the physical layer device is requested to internally loop the serialized transmit data to the de-serializer, and keep its transmit outputs static. Low for normal operation.

LCK_REF#

Output used to cause the SERDES device to lock the PLL to REFCLK.

COM_DET

Comma detect input. Indicates that data byte 0 of the first word contains a Comma character.

EN_CDET

Output. Used for enabling/disabling the byte alignment function of the external physical layer receiver. Enabled when high, disabled when low.

12.1.6 Signals used for SERDES and Serial Interface modes

REFCLK

Input clock. Nominal frequency is 125MHz and tolerance is +/- 100 ppm. REFCLK is the CORE Clock source of the Cassini, and it derives TXCLK_S in SERDES mode.

SIG_DET

Signal detect input. Single-ended PECL input. Driven by the optical receiver. A logic “1” indicates normal optical input levels, a logic “0” indicates a low optical input level fault. Used by Cassini+ to report “Remote Fault” to the other end of the link. This signal is used in SERDES and Serialink modes. Should be driven to a 3.3 V PECL logic “1” if the receiver does not support it (i.e. when using 8B/10B over copper).

VREF

Voltage reference input for single-ended PECL level. Should be externally connected to a 2.0V +/- 5% reference. This reference is only used to define the SIG_DET input threshold.

12.1.7 Hot Swap Signals

HS64EN#(CompactPCI connector pin)

Input. Hot Swap 64 Bit enable. A low state indicates back-plane is 64 bits wide..

BD_SEL#(CompactPCI connector pin)

Input signal. Board Slot Control. When lowl indicates its PCI Hot Swap card is fully inserted; back-end power can be turned on now.

ENUM#(CompactPCI connector pin)

Open Drain output. When low sends interrupt to system indicating its PCI Hot Swap card newly inserted or about to be extracted.

HEALTHY#(CompactPCI connector pin)

Open Drain output. When low indicates back-end power has reached acceptable level.

EJECT_SW

Ejector switch input.

BL_LED

Drives blue LED.

VPSRC_{L, R, T1, T2}

Inputs. Pre-charge bias voltage sources. A low impedance 1.0V bias.

DB_BD_SEL#

Input signal. Debonunce Board Slot Control.

BD_HEALTHY

Input. It is driven by the HEALTHY# ouput of a Power Sequencer to signal Cassini+ the power condition.

BD_PWR_ON

Output. It is asserted high when BD_SEL# is aserted. It can be used for driving a power FET swith if needed.

BD_SIG_ON

This output can possibly enable FET switches if needed to isolate the ASIC from board components which need protection from Cassini+ outputs when they are not powered but Cassini+ is.

PWR_FAULT#

Input. A power sequencer might have sensors to detect when excess current of some other problem has developed in the back-end power realm and drives this signal low..

HOT_SWAP

If Cassini+ is used in a Hot Swap board product, drive this input high and low otherwise. If high, the PCI bus interface wil use HS64EN# to determine if the system bus is 64 bits wide. If low, it uses the conventional sampling of REQ64# when PCI_RST# is desserted to determine bus width..

CHK_PLL

Input. When Cassini+ is set to be HotSwap and PCI 66MHz mode, If it is high, it resets internal to the chip, externally driven by the chip, and LOCAL_PCI_RST# as described in the Hot Swap spec. It is not desserted until the PLL indicates it has locked to the incoming PCI_CLOCK.

12.1.8 FCode PROM Interface Signals

P_A[15:0]

Outputs. PROM/Local_Bus address lines.

P_CS#

Output. Chip Select line to the PROM/Local Bus device.

P_OE#

Output. output enable to PROM/Local Bus device .

P_D[7:0]

Bidirection. PROM/Local Bus data lines.

RDWR#

Output. Local Bus Device R/W.

CSEXT#

Output. Chip Select line to Local Bus Device.

LB_SFT1

Output. Local Bus Soft Pin 1.Programable Local Bus control signal.

LB_SFT2

Output. Local Bus Soft Pin 2.Programable Local Bus control signal

12.1.9 Network Interface Diagnostic LEDs

LNKLED#

Link LED output. Active low.

FDPLXLED#

Full Duplex LED output. Active low.

TXLED#

Transmit activity LED output. Active low.

RXLED#

Receive activity LED output. Active low.

COLLED#

Collision LED output. Active low.

12.1.10 Testability

IDDTN#

Input used for IDD test during device manufacturing. Active low. Tie high for normal functional operation.

TX_TP1

Outputs. TX block test points.

RX_TP{1, 2}

Outputs. RX block test points..

ATPG_SI{1, 2, 3, 4}

Inputs uses for ATPG internal scan chains

ATPG_SO{1, 2, 3, 4}

Outputs used for ATPG internal scan chains.

12.1.11 JTAG Signals

TCLK

TDI

TDO

TMS

TRST#

12.2 Pin Assignment

Cassini+ pins fall into one of the following groups:

1. PCI Interface - 93 pins
2. PCI PLL - 5 pins
3. GMII/SERDES interfaces - 39 pins
4. Hot Swap Signals - 17 pins
5. PROM/Local Bus Interface - 30 pins
6. Network Diagnostic LEDs - 5 pins
7. Test - 17pins
8. Vdd - 28 pins
9. Vdd2 - 12 pins
10. PCIVio - 10 pins
11. GND - 44 pins

Total = 300 pins

The “Pinout Table” shown below provides the physical pin assignment for the Cassini+ ASIC. The TYPE column defines the drive characteristics. All I/Os use 3.3V supplies and signals that are 5V tolerant are identified as 5V compatible. The PCIVio supplies are meant to be connected to the Vi/o supply of the PCI slot for universal PCI card design.

Table 12-1 Pinout Table

INTERFACE	PIN NAME	BALL #	FUNCTION	TYPE
PCI	PCI_CLK	K19	PCI clock	input 5V compat.
PCI	RST#	J20	PCI reset	input 5V compat.
PCI	AD0	B8	address/data	bi-directional 5V compat.
PCI	AD1	D9	address/data	bi-directional 5V compat.
PCI	AD2	C9	address/data	bi-directional 5V compat.
PCI	AD3	B9	address/data	bi-directional 5V compat.

Table 12-1 Pinout Table

INTERFACE	PIN NAME	BALL #	FUNCTION	TYPE
PCI	AD4	E10	address/data	bi-directional 5V compat.
PCI	AD5	D10	address/data	bi-directional 5V compat.
PCI	AD6	C10	address/data	bi-directional 5V compat.
PCI	AD7	B11	address/data	bi-directional 5V compat.
PCI	AD8	C11	address/data	bi-directional 5V compat.
PCI	AD9	D11	address/data	bi-directional 5V compat.
PCI	AD10	E11	address/data	bi-directional 5V compat.
PCI	AD11	C12	address/data	bi-directional 5V compat.
PCI	AD12	D12	address/data	bi-directional 5V compat.
PCI	AD13	A13	address/data	bi-directional 5V compat.
PCI	AD14	B13	address/data	bi-directional 5V compat.
PCI	AD15	D13	address/data	bi-directional 5V compat.
PCI	AD16	D16	address/data	bi-directional 5V compat.
PCI	AD17	C17	address/data	bi-directional 5V compat.
PCI	AD18	D18	address/data	bi-directional 5V compat.
PCI	AD19	D19	address/data	bi-directional 5V compat.

Table 12-1 Pinout Table

INTERFACE	PIN NAME	BALL #	FUNCTION	TYPE
PCI	AD20	E17	address / data	bi-directional 5V compat.
PCI	AD21	E18	address / data	bi-directional 5V compat.
PCI	AD22	E19	address / data	bi-directional 5V compat.
PCI	AD23	E20	address / data	bi-directional 5V compat.
PCI	AD24	F20	address / data	bi-directional 5V compat.
PCI	AD25	G16	address / data	bi-directional 5V compat.
PCI	AD26	G17	address / data	bi-directional 5V compat.
PCI	AD27	G18	address / data	bi-directional 5V compat.
PCI	AD28	G19	address / data	bi-directional 5V compat.
PCI	AD29	H17	address / data	bi-directional 5V compat.
PCI	AD30	H18	address / data	bi-directional 5V compat.
PCI	AD31	H19	address / data	bi-directional 5V compat.
PCI	AD32	L3	address / data	bi-directional 5V compat.
PCI	AD33	L2	address / data	bi-directional 5V compat.
PCI	AD34	K2	address / data	bi-directional 5V compat.
PCI	AD35	K3	address / data	bi-directional 5V compat.

Table 12-1 Pinout Table

INTERFACE	PIN NAME	BALL #	FUNCTION	TYPE
PCI	AD36	K4	address/data	bi-directional 5V compat.
PCI	AD37	K5	address/data	bi-directional 5V compat.
PCI	AD38	J1	address/data	bi-directional 5V compat.
PCI	AD39	J2	address/data	bi-directional 5V compat.
PCI	AD40	J3	address/data	bi-directional 5V compat.
PCI	AD41	H1	address/data	bi-directional 5V compat.
PCI	AD42	H2	address/data	bi-directional 5V compat.
PCI	AD43	H3	address/data	bi-directional 5V compat.
PCI	AD44	H4	address/data	bi-directional 5V compat.
PCI	AD45	G2	address/data	bi-directional 5V compat.
PCI	AD46	G3	address/data	bi-directional 5V compat.
PCI	AD47	G4	address/data	bi-directional 5V compat.
PCI	AD48	G5	address/data	bi-directional 5V compat.
PCI	AD49	F1	address/data	bi-directional 5V compat.
PCI	AD50	F3	address/data	bi-directional 5V compat.
PCI	AD51	F4	address/data	bi-directional 5V compat.

Table 12-1 Pinout Table

INTERFACE	PIN NAME	BALL #	FUNCTION	TYPE
PCI	AD52	E1	address / data	bi-directional 5V compat.
PCI	AD53	E2	address / data	bi-directional 5V compat.
PCI	AD54	E3	address / data	bi-directional 5V compat.
PCI	AD55	E4	address / data	bi-directional 5V compat.
PCI	AD56	D2	address / data	bi-directional 5V compat.
PCI	AD57	D3	address / data	bi-directional 5V compat.
PCI	AD58	C4	address / data	bi-directional 5V compat.
PCI	AD59	D5	address / data	bi-directional 5V compat.
PCI	AD60	C5	address / data	bi-directional 5V compat.
PCI	AD61	B5	address / data	bi-directional 5V compat.
PCI	AD62	A5	address / data	bi-directional 5V compat.
PCI	AD63	D6	address / data	bi-directional 5V compat.
PCI	C / BE[3]# C / BE[2]# C / BE[1]# C / BE[0]#	F17 C16 C13 B10	command / byte enable for lower 32 bits	bi-directional 5V compat.
PCI	C / BE[7]# C / BE[6]# C / BE[5]# C / BE[4]#	A8 C7 E7 A6	command / byte enable for upper 32 bits	bi-directional 5V compat.
PCI	PAR	B15	parity for lower 32 bits	bi-directional 5V compat.

Table 12-1 Pinout Table

INTERFACE	PIN NAME	BALL #	FUNCTION	TYPE
PCI	PAR64	C6	parity for upper 32 bits	bi-directional 5V compat.
PCI	FRAME#	A15	cycle frame	bi-directional 5V compat.
PCI	IRDY#	B16	initiator ready	bi-directional 5V compat.
PCI	TRDY#	C15	terminator ready	bi-directional 5V compat.
PCI	STOP#	B14	stop	bi-directional 5V compat.
PCI	IDSEL	F18	initialization device select	input 5V compat.
PCI	DEVSEL#	D15	device select	bi-directional 5V compat.
PCI	REQ#	H20	request	output 5V compat.
PCI	REQ64#	E8	request 64 bit transfer	output 5V compat.
PCI	GNT#	J18	grant	input 5V compat.
PCI	PERR#	C14	parity error	bi-directional 5V compat.
PCI	SERR#	E14	system error	output 5V compat.
PCI	INTA#	K16	interrupt A	open drain output 5V compat.
PCI	INTB#	V9	interrupt B	ditto
PCI	INTC#	W9	interrupt C	ditto
PCI	INTD#	Y9	interrupt D	ditto
PCI	ACK64#	B7	acknowledge for 64 bit transfer	input 5V compat.

Table 12-1 Pinout Table

INTERFACE	PIN NAME	BALL #	FUNCTION	TYPE
PCI	M66EN	A12	66MHz Enable line	input 5V compat.
PCI PLL	PLL_BYPASS#	K17	PCI clock PLL bypass	input
PCI PLL	PLL_CREF	L18	PLL CAP	input
PCI PLL	PLL_CK_OUT	N1	PLL Clock Test point	output
PCI PLL	PLL_VDD	L19		supply
PCI PLL	PLL_VSS	K18		supply
GMII/ SERDES	CRS	W15	MII: CRS (Carrier) GMII: CRS (Carrier) SERDES: N.C.	input 5V compat.
GMII/ SERDES	COL	U14	MII: COL (Collision) GMII: COL (Collision) SERDES: N.C.	input 5V compat.
GMII/ SERDES	RXCLK	T13	MII: RXCLK (receive clock) GMII: RXCLK (receive clock) SERDES: N.C.	input 5V compat.
GMII/ SERDES	RBC0	T14	MII: N.C. GMII: N.C. SERDES:RBC 0 (receive byte clock phase1)	input 5V compat.

Table 12-1 Pinout Table

INTERFACE	PIN NAME	BALL #	FUNCTION	TYPE
GMII/ SERDES	RBC1	Y15	MII: N.C. GMII: N.C. SERDES:RBC 1 (receive byte clock phase2)	input 5V compat.
GMII/ SERDES	RX3_S RX2_S RX1_S RX0_S	U12 V12 W12 Y12	MII: RXD[3:0] receive data GMII: RXD[3:0] receive data SERDES: RX[3:0]	input 5V compat.
GMII/ SERDES	RX7_S RX6_S RX5_S RX4_S	U13 V13 W13 Y13	MII: N.C. GMII: RXD[7:4] receive data SERDES: RX[7:4]	input 5V compat.
GMII/ SERDES	RX8_S	W14	MII: RXDV (receive data valid) GMII: RXDV (receive data valid) SERDES: RX8	input 5V compat.
GMII/ SERDES	RX9_S	V14	MII: RXER (receive error) GMII: RXER (receive error) SERDES: RX9	input 5V compat.
GMII/ SERDES	TXCLK_MII	P19	MII: TXCLK (transmit clock) GMII: N.C. SERDES: N.C.	input 5V compat.

Table 12-1 Pinout Table

INTERFACE	PIN NAME	BALL #	FUNCTION	TYPE
GMII/ SERDES	TX8_S	R20	MII: TXEN (transmit enable) GMII: TXEN (transmit enable) SERDES: TX8	output 5V compat.
GMII/ SERDES	TX9_S	P16	MII: TXER (transmit error) GMII: TXER (transmit error) SERDES: TX9	output 5V compat.
GMII/ SERDES	TX3_S TX2_S TX1_S TX0_S	T19 T18 T17 U19	MII: TXD[3:0] transmit data GMII: TXD[3:0] transmit data SERDES: TX[3:0]	output 5V compat.
GMII/ SERDES	TX7_S TX6_S TX5_S TX4_S	R19 R18 R17 T20	MII: N.C. GMII: TXD[7:4] transmit data SERDES: TX[7:4]	output 5V compat.
GMII/ SERDES	TXCLK_S	V15	MII: N.C. GMII: TXCLK_OUT SERDES: TBC (Transmit byte clock)	output
GMII/ SERDES	MDC	V11	MII: MDC (management clock) GMII: optional SERDES: N.C.	output 5V compat.

Table 12-1 Pinout Table

INTERFACE	PIN NAME	BALL #	FUNCTION	TYPE
GMII/ SERDES	MDIO0	W10	MII: MDIO0 (management data) GMII: optional SERDES: N.C.	bi-directional 5V compat.
GMII/ SERDES	MDIO1	W11	MII: MDIO1 (management data) GMII: optional SERDES: N.C.	bi-directional 5V compat.
GMII/ SERDES	MII_BUF_EN	V10	MII Buffer Enable	output 5V compat
GMII/ SERDES	RSTOUT#	U18	Reset Output	output 5V compat
SERDES	EWRAP	Y16	Wrap control output	output 5V compat
SERDES	COM_DET	U15	Comma character detection indication	input 5V compat.
SERDES	EN_CDET	V16	Comma character detection enable	output 5V compat
SERDES	LCK_REF#	W16	Lock to Reference	output
Serial/SERDES	REFCLK	V17	Reference clock	input 5V compat.
Serial/SERDES	SIG_DET	U16	Signal detection indication from optical receiver	single-ended PECL input

Table 12-1 Pinout Table

INTERFACE	PIN NAME	BALL #	FUNCTION	TYPE
Serial / SERDES	VREF	W17	Reference voltage for single-ended PECL level. Externally connected to 2.0 V.	single-ended PECL input
PCI HotSwap	HS64EN#	B6	Hotswap 64bit enable	input 5V compat.
PCI HotSwap	BD_SEL#	A16	Board Select	input 5V compat.
PCI HotSwap	ENUM#	C8	Board plug in / out indicator	Open Drain Output
PCI HotSwap	HEALTHY#	J19	Board Healthy	Open Drain Outpu
HotSwap	EJECT_SW	L4	Ejector Switch	input 5 V compat
HotSwap	BL_LED	L5	Status LED	Output
HotSwap	VPSRC_L VPSRC_R VPSRC_T1 VPSRC_T2	H5 H16 E13 D8	Pre-charge bias voltage source	Input 1.0 V bias
HotSwap	DB_BD_SEL#	N19	Debounce Board Select	input 5 V compat
HotSwap	BD_HEALTHY	N20	Board Power Healthy	intput 5 V compat
HotSwap	BD_PWR_ON	N18	Back-end power on	Output
HotSwap	BD_SIG_ON	N17	Back-end signal on	Output
HotSwap	PWR_FAULT#	M17	Power status	input 5 V compat

Table 12-1 Pinout Table

INTERFACE	PIN NAME	BALL #	FUNCTION	TYPE
HotSwap	HOT_SWAP	P18	HotSwap on	input, active high
HotSwap	CHK_PLL	N16	Check PLL lock	input, active high
PROM / Local Bus	P_A0	T4	address line	output 5 V compat
PROM / Local Bus	P_A1	T3	address line	output 5 V compat
PROM / Local Bus	P_A2	T2	address line	output 5 V compat
PROM / Local Bus	P_A3	T1	address line	output 5 V compat
PROM / Local Bus	P_A4	R4	address line	output 5 V compat
PROM / Local Bus	P_A5	R3	address line	output 5 V compat
PROM / Local Bus	P_A6	R2	address line	output 5 V compat
PROM / Local Bus	P_A7	R1	address line	output 5 V compat
PROM / Local Bus	P_A8	P5	address line	output 5 V compat
PROM / Local Bus	P_A9	P4	address line	output 5 V compat
PROM / Local Bus	P_A10	P3	address line	output 5 V compat
PROM / Local Bus	P_A11	P2	address line	output 5 V compat
PROM / Local Bus	P_A12	N5	address line	output 5 V compat
PROM / Local Bus	P_A13	N4	address line	output 5 V compat

Table 12-1 Pinout Table

INTERFACE	PIN NAME	BALL #	FUNCTION	TYPE
PROM / Local Bus	P_A14	N3	address line	output 5 V compat
PROM / Local Bus	P_A15	N2	address line	output 5 V compat
PROM / Local Bus	P_D0	Y6	data bus	input 5V compat.
PROM / Local Bus	P_D1	W6	data bus	input 5V compat.
PROM / Local Bus	P_D2	V6	data bus	input 5V compat.
PROM / Local Bus	P_D3	U6	data bus	input 5V compat.
PROM / Local Bus	P_D4	Y5	data bus	input 5V compat.
PROM / Local Bus	P_D5	W5	data bus	input 5V compat.
PROM / Local Bus	P_D6	V5	data bus	input 5V compat.
PROM / Local Bus	P_D7	U5	data bus	input 5V compat.
PROM / Local Bus	P_CS#	U2	PROM chip select	output 5 V compat
PROM / Local Bus	P_OE#	U3	output enable	output 5 V compat
PROM / Local Bus	RDWR#	V4	Local Bus W / R	output 5 V compat
PROM / Local Bus	CSEXT#	W4	Local Bus Chip Select	output 5 V compat
PROM / Local Bus	LB_SFT1	M1	Programable Soft Pin	output 5 V compat
PROM / Local Bus	LB_SFT2	M2	Programable Soft Pin	output 5 V compat

Table 12-1 Pinout Table

INTERFACE	PIN NAME	BALL #	FUNCTION	TYPE
LEDs	LNKLED#	T8	Link LED	open drain
LEDs	FDPLXLED#	U8	Full Duplex LED	open drain
LEDs	TXLED#	V8	Transmit activity LED	open drain
LEDs	RXLED#	W8	Receive activity LED	open drain
LEDs	COLLED#	Y8	Collision LED	open drain
TESTING	IDDTN#	P17	IDDT	input active low
TESTING	TX_TP1	U9	TX block TP	output
TESTING	RX_TP1	V7	RX block TP	output
TESTING	RX_TP2	W7	RX block TP	output
TESTING	ATPG_SI1	M4	Scan Chain 1	input
TESTING	ATPG_SI2	U7	Scan Chain 2	input
TESTING	ATPG_SI3	T10	Scan Chain 3	input
TESTING	ATPG_SI4	T11	Scan Chain 4	input
TESTING	ATPG_SO1	M3	Scan Chain 1	output
TESTING	ATPG_SO2	T7	Scan Chain 2	output
TESTING	ATPG_SO3	U10	Scan Chain 3	output
TESTING	ATPG_SO4	U11	Scan Chain 4	output
JTAG	TCLK	M18	JTAG clock	input
JTAG	TDI	L17	JTAG data in	input

Table 12-1 Pinout Table

INTERFACE	PIN NAME	BALL #	FUNCTION	TYPE
JTAG	TDO	L16	JTAG data out	output
JTAG	TMS	M19	JTAG test mode select	input
JTAG	TRST#	M20	JTAG reset	input
Supplies	PCIVIO	F2, J4 J17, F19 B17,D14 B12, A9 D7,B4	PCI buffers VDD from I/O pins (can be 5V or 3.3V)	

Table 12-1 Pinout Table

INTERFACE	PIN NAME	BALL #	FUNCTION	TYPE
Supplies	VDD	A10 A11 A17 A19 A2 A4 B1 B20 D1 D17 D20 D4 K1 K20 L1 L20 U1 U17 U20 U4 W1 W20 Y10 Y11 Y17 Y19 Y2 Y4	IO VDD	3.3V
Supplies	VDD2	E15 E16 E5 E6 F16 F5 R16 R5 T15 T16 T5 T6	CORE VDD	2.5V

Table 12-1 Pinout Table

INTERFACE	PIN NAME	BALL #	FUNCTION	TYPE
Supplies	VSS	A1 A14 A20 A7 B19 B2 C18 C3 E12 E9 G1 G20 J10 J11 J12 J16 J5 J9 K10 K11 K12 K9 L10 L11 L12 L9 M10 M11 M12 M16 M5 M9 P1 P20 T12 T9 V18 V3 W19 W2 Y1 Y14 Y20 Y7	VSS	

12.3 Electrical Specifications

12.3.1 Absolute Maximum ratings

Table 12-2 Absolute maximum ratings

Symbol	Parameter	Limit	Unit
VDD	IO DC Power Supply Voltage	-0.3 to 3.9	V
VDD2	Core DC Power Supply Voltage	-0.3 to 3.2	V
Vin, Vout	DC Input, Output Voltage	VSS - 0.3 to VDD + 0.3	V
Vin	DC Input Voltage for 5V compatible inputs	-1.0 to 6.5	V
Tstg	Storage Temperature	-40 to 125	DegC
Pd	Power dissipation	TBD	Watts

Note – Stresses beyond those listed in the above table may cause physical damage to the device and should be avoided.

12.3.2 Recommended Operating Conditions

Table 12-3 Recommended Operating Conditions

Symbol	Parameter	Limit	Unit
VDD	IO DC Power Supply Voltage	3.3 +/- 5%	V
VDD2	Core DC power Supply Voltage	2.5 +/- 5%	
Vin, Vout	DC Input, Output Voltage	0 to VDD	V
Tj	Junction Temperature	0 to 105	DegC
Ta	Operating Ambient Temperature	0 to 65	DegC

12.3.3 DC Characteristics

Table 12-4 DC Specification for PCI Interface

Symbol	Parameter	Conditions	Min	Max	Unit
Vil	Input Low Voltage		0.5*VDD	VDD+0.5	V
Vih	Input High Voltage		-0.5	0.3*VDD	V
Iil	Input Leakage Current Low	0<Vin<Vss	-10	10	uA
Vol	Output Low Voltage	Iol=1500 uA		0.1*VDD	V
Voh	Output High Voltage	Ioh=-500 uA	0.9*VDD		V
Cout	Output Pin Capacitance	excludes package		10	pF

Table 12-5 DC Specifications of PCS/GMII, EPROM, JTAG, LED, MDIO Interfaces

Symbol	Parameter	Conditions	Min	Max	Unit
Vil	Input Low Voltage		Vss-0.5	0.8	V
Vih	Input High Voltage		2.0	6.5	V
Iil	Input Leakage Current	Vin = 0 to VDD	- 10	10	uA
Vt	Schmitt-trigger, Input Switching Threshold Voltage			2.0	V
Vt+	Schmitt-trigger, Input Switching Threshold Voltage, rising edge			2.0	V
Vt-	Schmitt-trigger, Input Switching Threshold Voltage, falling edge			1.0	V
Cin	Input Capacitance	excludes package		3.0	pF
Vol	Output Low Voltage		0	0.4	V
Voh	Output High Voltage		2.4	VDD	V
Ioh	Output High Current	bt6f@VOL=0.4V		13.25	mA
Iol	Output Low Current	bt6f@VOL=0.4V	4	9.6	mA
Ioz	High Z Leakage current (open drain)	Vpad=5.5V	-10	10	uA
Cout	Output Capacitance	excludes package		3	pF

12.3.4 Environmental ELectrical Protection

Table 12-6 Environmental Electrical Protection

ESD	Latch UP
Minimum	Minimum
2KV	150 mA

12.3.5 AC Characteristics

12.3.5.1 PCI Interface

Table 12-7 PCI Clock AC Timing Specifications

Symbol	Parameter	66 MHz Min	66 MHz Max	33 MHz Min	33 MHz Max	Unit
T_cyc	pci_clk Cycle Time	15	30	30	DC	ns
T_high	pci_clk High Time	6		11		ns
T_low	pci_clk Low Time	6		11		ns

Table 12-8 PCI Input AC Timing Characteristics

Symbol	Parameter	66 MHz Min	66 MHz Max	33 MHz Min	33 MHz Max	Unit
T_su	Input Setup Time to bused inputs	3		7		ns
T_su(ptp)	Input Setup Time to pci_req_l, pci_gnt_l	5		10,12		ns
T_hold	Input Hold Time from pci_clk	0		0		ns
T_rst	Reset Active Time after Power Stable	1		1		ms
T_rst(clk)	Reset Active after pci_clk stable	100		100		us
T_rst(off)	Reset Active to Output Float Delay		40		40	ns

Input signals are PCI_CLK, RST#, IDSEL, GNT#. These signals except RST# are synchronous to the positive edge of PCI_CLK. RST# assertion and de-assertion is asynchronous toPCI_CLK.

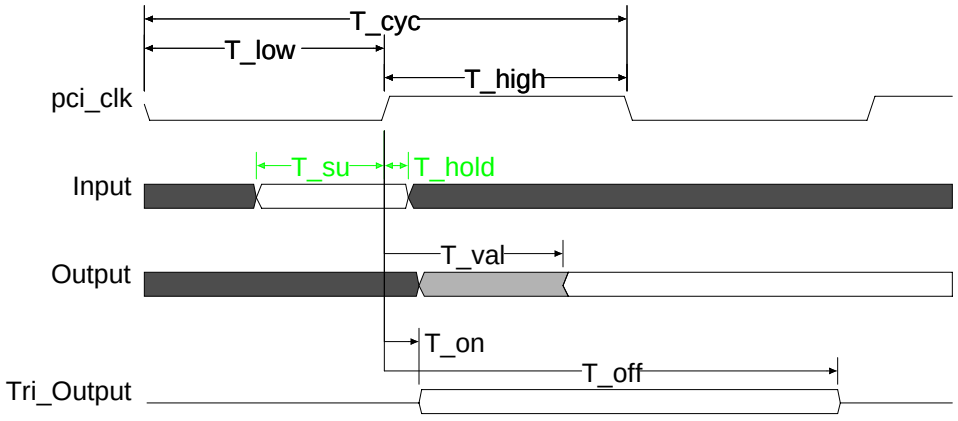
Table 12-9 PCI Output AC Timing Characteristics

Symbol	Parameter	66 MHz Min	66 MHz Max	33 MHz Min	33 MHz Max	Unit
T_val	pci_clk to Signal Valid Delay - bused	2	6	2	11	ns
T_val(ptp)	pci_clk to Signal Valid Delay - pci_req_l, pci_gnt_l	2	6	2	12	ns
T_on	Float to Active Delay	2		2		ns
T_off	Active to Float Delay		14		28	ns
T_rrsu	REQ# to RST# setup time	10*T_cyc		10*T_cyc		us
T_rrh	REQ# to RST# hold time	0	50	0	50	ns

PCI output signals are REQ#, INTA#. The assertion and de-assertion of INTA# is asynchronous to PCI_CLK.

PCI bi-directional signals are AD[63:0], C/BE[7:0], LOCK#, PAR, FRAME#, IRDY#, PERR#, STOP#, PAR64#, REQ64#, ACK64#. These signals must meet the timing requirements mentioned in the PCI Input and PCI Output Timing Characteristics tables.

Figure 12-1 PCI Timing Waveform



12.3.5.2 PCS Interface

Table 12-10 PCS Receive Bus AC Specification

Symbol	Parameter	125 MHz Min	125 MHz Typ	125 MHz Max	Unit
Tfreq	RX_CLK Frequency		62.5		MHz
Tdrift	RX_CLK Drift Rate	0.2			us/MHz
Tsetup	RX_S data bus, COM_DET, LOCKREFN setup to RX_CLK	2.5			ns
Thold	RX_S data bus, COM_DET, LOCKREFN hold from RXCLK	1.5			ns
Tduty	RXCLK duty cycle	40		60	%
Ta-b	RXCLK Skew	7.5		8.5	ns
Tdo	EN_CDET setup to RXCLK	2.0			ns

Figure 12-2 PCS Receive Interface Timing Waveforms

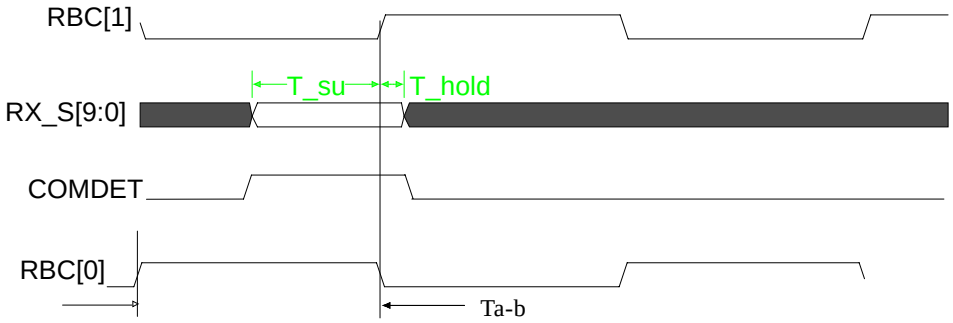


Table 12-11 PCS Transmit Bus AC Specification

Symbol	Parameter	125 MHz Min	125 MHz Typ	125 MHz Max	Unit
Tperiod_ref	REFCLK Period, +/- 100 ppm tolerance		8		ns
Tperiod	TXCLK_S Period, +/- 100 ppm tolerance		8		ns
Tsetup	TX_S data bus setup to TXCLK_S	2.0			ns
Thold	TX_S data bus hold from TXCLK_S	1.0			ns
Tduty	TXCLK_S duty cycle	40		60	%

12.3.5.3 GMII Interface

*Table 12-12*GMII General AC Specifications

Symbol	Parameter	Conditions	Min	Max	Units
Tr	Clock rise time	Vil_ac(max) to Vih_ac(min)		1.00	ns
Tsetup	Clock fall time	Vil_ac(min) to Vih_ac(max)		1.00	ns
-	Clock Slew Rate (rising)	Vil_ac(max) to Vih_ac(min)	0.6		V / ns
-	Clock Slew Rate (falling)	Vil_ac(min) to Vih_ac(maxn)	-0.6		V / ns
Vil_ac	Input Low Voltage AC			0.70	V
Vih_ac	Input High Voltage		1.90		V

*Table 12-13*GMII AC Specifications for Transmit signals

Symbol	Parameter	Min	Max	Units
Tperiod	TXCLK_S Period	7.50		ns
Thigh	TXCLK_S Time High	2.50		ns
Tlow	TXCLK_S Time Low	2.50		ns
Tsetup	TXD, TX_EN, TX_ER Setup to rising edge of TXCLK_S	2.50		ns
Thold	TXD, TX_EN, TX_ER Hold fromrising edge of TXCLK_S	0.50		ns

*Table 12-14*GMII AC Specifications or Receive signals

Symbol	Parameter	Min	Max	Units
Tperiod	RXCLK Period	7.50		ns
Thigh	RXCLK Time High	2.50		ns
Tlow	RXCLK Time Low	2.50		ns
Tsetup	RXD, RX_DV, RX_ER Setup to RXCLK	2.00		ns
Thold	RXD, RX_DV, RX_ER Hold to RXCLK	0.00		ns

Note – CRS and COL signals are asynchronous in MII and GMII Interfaces.

*Table 12-15*MDIO / MDC AC Specification

Symbol	Parameter	Min	Max	Units
Tperiod	MDC Period	400		ns
Thigh	MDC Time High	160		ns
Tlow	MDC Time Low	160		ns

Tsetup (driven by Cassini+)	MDIO to MDC, when MDIO is being driven by Cassini+	10		ns
Thold (driven by Cassini+)	MDIO from MDC, when MDIO is being driven by Cassini+	10		ns
Tsetup (driven by PHY)	MDIO to MDC, when MDIO is being driven by PHY	100		ns
Thold (driven by PHY)	MDIO from MDC, when MDIO is being driven by PHY	0		ns

*Table 12-16*EPROM AC Specification

Symbol	Parameter	Min	Max	Units
Tsetup	P_A bus Setup to PCICLK	7.00		ns
Thold	P_A bus Hold to PCICLK	2.00		ns
Tdo	PCICLK to P_D data bus valid	1.0	7.0	ns

*Table 12-17*JTAG AC Specification

Symbol	Parameter	Min	Max	Units
Tclk	TCLK	2	10	MHz
Tsetup	TDI, TMS setup to Jtag TCLK	5		ns
Thold	TDI, TMS hold to Jtag TCLK	10		ns
Tdo	TDO output delay from Jtag TCLK	TCLK/2 + 7.49	TCLK/2 + 15.90	ns
Trst_	TRST_ assertion time	5 TClk		

12.4 Mechanical Information

12.4.1 Package info & drawings

Cassini+ 's package is a four layer 316 pin PBGA. Figure 12-3 is a top view of the package solder mask.

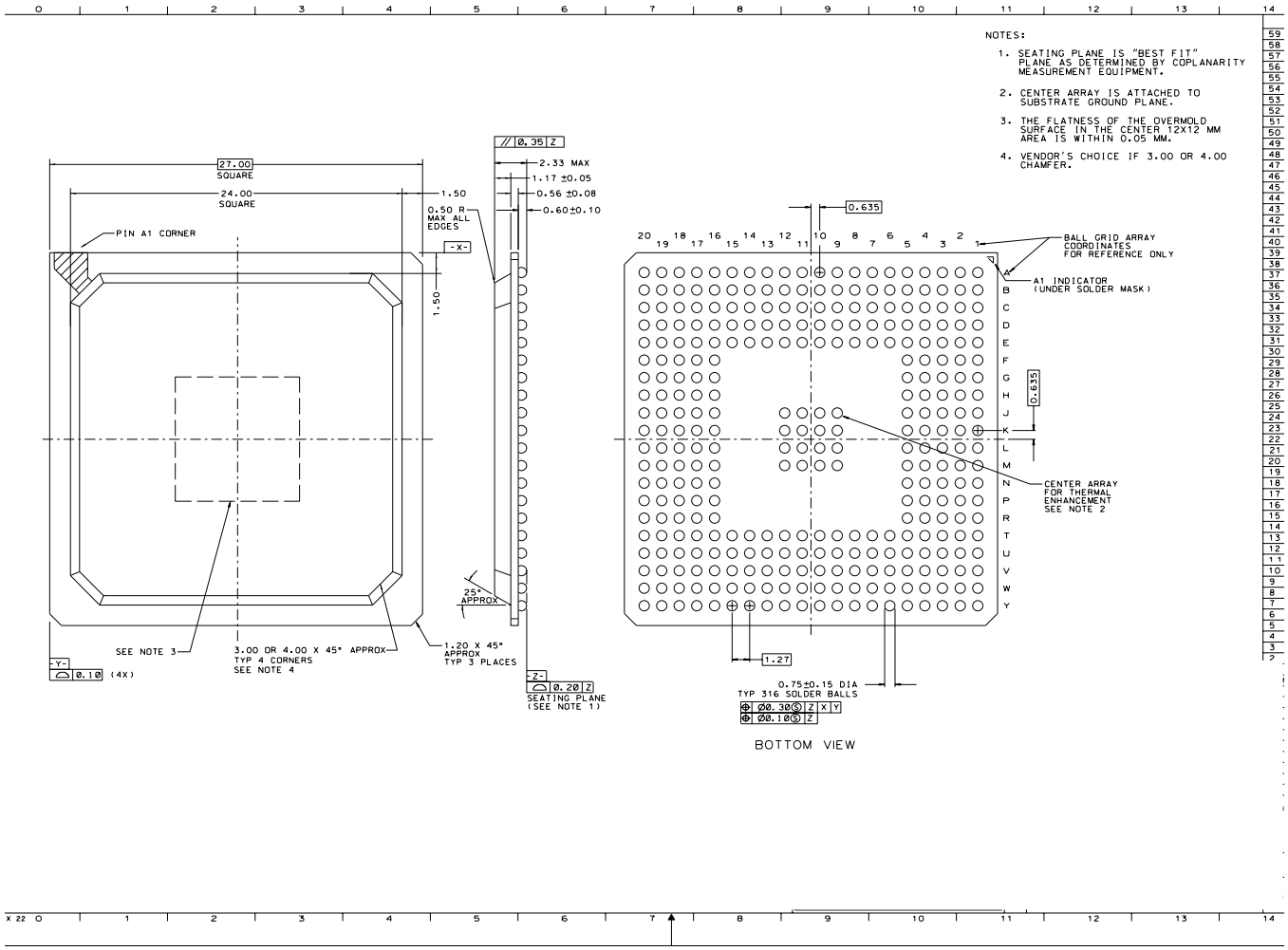


Figure 12-3 Pacakge footprint

12.4.2 Package characteristics

*Table 12-18*s Pin and Size Characteristics

Package type	Pin Count	Cavity Size	Signals	Die Range
316 PBGAM	316	27x27x2.56mm	232	9.3 mm max.

Table 12-19 Thermal Resistance

no heat sink	0 lfpm	200 lfpm	400 lfpm	600 lfpm
Theta Ja (Degrees C / Watt)	21	18.5	17.5	16.5

Theta Jc (Degrees C / Watt)	3.0

Table 12-20 Pin Loading

	Min	Max	Unit
Resistance	230	288	mO
Inductance	4.42	6.75	nH
Capacitance	0.72	0.92	pF
Crosstalk Capacitance	0.08	0.12	pF